修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報システム学研究科 情報ネットワークシステム学専攻　博士前期課程		
氏　　名	鮎川　俊	学籍番号	1252001
論 文 題 目	モニタリングに基づく TCP 輻輳制御方式の推定に関する研究		

要　旨

近年，インターネットはコンテンツの多様化につれ，コンテンツの大容量化が進んでいる．それに伴いネットワークを流れるトラフィックは日々増大している．このようなネットワークに対応するためには，ネットワーク帯域を公平かつ効率的に利用することが重要であり，様々な観点からトラフィックの傾向を把握する必要がある．現在，ネットワークを流れるトラフィックには TCP (Transmission Control Protocol)プロトコルが広く用いられている．TCP には，現在まで複数の輻輳制御アルゴリズムが提案されており，その輻輳制御アルゴリズムが通信に大きな影響を与えると考えられる. また利己的な挙動をするように TCP を改造する悪意を持ったユーザーも存在する．このようなユーザーが増加すると通常の TCP は十分な性能が得られなくなる．このためネットワーク管理者は，ネットワークを流れる TCP トラフィックを監視し，それぞれの通信フローにおける TCP アルゴリズムを推定する技術が重要であると考えられる．本研究では，ネットワークをモニタリングし，TCP のデータや ACK の送信状況等を記録している通信ログから，クライアント側の輻輳ウィンドウの変化を推定し, さらにその結果に基づいて TCP バージョンの推定を行う方法を提案する．

通信ログにはサーバ・クライアント間で行われている通信の詳細が記録されており，TCP プロトコルを使用している通信には通信時間やデータの送信シーケンス番号, ACK 番号等が明記されている．このデータログを用いて輻輳ウィンドウの推定を行う．

方法として，まず i 番目の ACK 応答を受信してから i+1 番目の ACK 応答を受信するまでの間に送信したデータの最大送信シーケンス番号から i 番目の ACK 番号を減算し，それを MSS (Maximum Segment Size)で割った数を輻輳ウィンドウの仮推定値としておく．その上で i 番目の ACK 応答の RTT (Round Trip Time)の間だけ監視し，その間に計測した輻輳ウィンドウの仮推定値の中で最も大きい輻輳ウィンドウの値を，RTT 測定開始時点での輻輳ウィンドウの推定値とする．

さらにこの推定値と，推定値間の輻輳ウィンドウの増加量を散布図としてプロットする．この散布図と，TCP の輻輳制御アルゴリズムの特徴から推測される輻輳ウィンドウの変化をグラフにして比較することで，TCP の輻輳制御アルゴリズムを推定していく．

平成２５年度修士論文

モニタリングに基づく
TCP輻輳制御方式の推定に関する研究

電気通信大学大学院情報システム学研究科

情報ネットワークシステム学専攻

学 籍 番 号：1252001

氏　　名：鮎川　俊

主任指導教員：加藤　聰彦　教授

指 導 教 員：大坐畠　智　准教授

指 導 教 員：岡田　和則　客員教授

提 出 年 月 日：平成２６年２月２１日（金）

概要

近年，インターネットはコンテンツの多様化につれ，コンテンツの大容量化が進んでいる．それに伴いネットワークを流れるトラフィックは日々増大している．このようなネットワークに対応するためには，ネットワーク帯域を公平かつ効率的に利用することが重要であり，様々な観点からトラフィックの傾向を把握する必要がある．現在，ネットワークを流れるトラフィックには TCP (Transmission Control Protocol)プロトコルが広く用いられている．TCP には，現在まで複数の輻輳制御アルゴリズムが提案されており，その輻輳制御アルゴリズムが通信に大きな影響を与えると考えられる．また利己的な挙動をするように TCP を改造する悪意を持ったユーザーも存在する．このようなユーザーが増加すると通常の TCP は十分な性能が得られなくなる．このためネットワーク管理者は，ネットワークを流れる TCP トラフィックを監視し，それぞれの通信フローにおける TCP アルゴリズムを推定する技術が重要であると考えられる．本研究では，ネットワークをモニタリングし，TCP のデータや ACK の送信状況等を記録している通信ログから，クライアント側の輻輳ウィンドウの変化を推定し，さらにその結果に基づいて TCP バージョンの推定を行う方法を提案する．

通信ログにはサーバ・クライアント間で行われている通信の詳細が記録されており，TCP プロトコルを使用している通信には通信時間やデータの送信シーケンス番号，ACK 番号等が明記されている．このデータログを用いて輻輳ウィンドウの推定を行う．

方法として，まず i 番目の ACK 応答を受信してから i+1 番目の ACK 応答を受信するまでの間に送信したデータの最大送信シーケンス番号から i 番目の ACK 番号を減算し，それを MSS (Maximum Segment Size)で割った数を輻輳ウィンドウの仮推定値としておく．その上で i 番目の ACK 応答の RTT (Round Trip Time)の間だけ監視し，その間に計測した輻輳ウィンドウの仮推定値の中で最も大きい輻輳ウィンドウの値を，RTT 測定開始時点での輻輳ウィンドウの推定値とする．

さらにこの推定値と，推定値間の輻輳ウィンドウの増加量を散布図としてプロットする．この散布図と，TCPの輻輳制御アルゴリズムの特徴から推測される輻輳ウィンドウの変化をグラフにして比較することで，TCPの輻輳制御アルゴリズムを推定していく．

目次

第 1 章　　はじめに

1.1　研究背景

近年，インターネットはコンテンツの多様化につれ，コンテンツの大容量化が進んでいる．それに伴いネットワークを流れるトラフィックは日々増大している．このようなネットワークに対応するためには，ネットワーク帯域を公平かつ効率的に利用することが重要であり，様々な観点からトラフィックの傾向を把握する必要がある．現在，ネットワークを流れるトラフィックには TCP(Transmission Control Protocol)プロトコルが広く用いられている．TCP には，現在まで複数の輻輳制御アルゴリズムが提案されており，その輻輳制御アルゴリズムが通信に大きな影響を与えると考えられる．また利己的な挙動をするように TCP を改造する悪意を持ったユーザーも存在する．このようなユーザーが増加すると通常の TCP は十分な性能が得られなくなる．このためネットワーク管理者は，ネットワークを流れる TCP トラフィックを監視し，それぞれの通信フローにおける TCP アルゴリズムを推定する技術が重要であると考えられる．本研究では，ネットワークをモニタリングし，TCP のデータや ACK の送信状況等を記録している通信ログから，クライアント側の輻輳ウィンドウの変化を推定し，さらにその結果に基づいて TCP バージョンの推定を行う方法を提案する．

1.2　本稿の構成

本稿は以降, 第２章でTCPに関する基本的な概要について説明する. 第３章ではTCPの輻輳制御方式とアルゴリズムについて説明する. 第４章では本研究の核となる提案方式について説明する. 第５章では提案に基づく実験と結果をまとめ, 第４章の提案方式を検証する.

第２章　　TCP の概要

本章では，本研究の対象である TCP プロトコルについての概要や基本的な知識について述べる．

2.1　TCP/IP 階層モデル

ネットワーク・プロトコルは一般的にレイヤという概念をもとに開発され，各階層は通信の異なる相に対して責任をもつ．TCP/IP などのプロトコル群は，さまざまな階層のプロトコルの組み合わせである．TCP/IP は通常，図 1 に示した 4 階層システムとして認識されている．[1]

アプリケーション層	HTTP，FTP，POP3，SMTP など
トランスポート層	TCP，UDP
インターネット層	IP，ICMP，IGMP
ネットワークインターフェース層	Ethernet，PPP

図 1．TCP/IP 階層モデルと主なプロトコル

ネットワークインターフェース層では，ケーブルとの物理的なインターフェースに関する全てのハードウェア的側面を制御する．代表的なプロトコルとして，イーサネット(Ethernet)や PPP (Point to Point Protocol)がある．

インターネット層では，ネットワークのパケットの移動を制御する．パケットのルーティングはこの階層で行われる．主なプロトコルに IP(Internet Protocol)，ICMP(Internet Control Message Protocol)，IGMP(Internet Group Management Protocol)がある．

トランスポート層では，上位のアプリケーション層に対して，2 台のホスト間のデータの流れを提供する．このプロトコルには TCP，UDP (User Datagram Protocol)がある．

アプリケーション層では，特定アプリケーションの詳細な動作を処理する．TCP/IP にはさまざまな共通のアプリケーションがあり，ほとんどすべての実装製品に用意されている．主なプロトコルとして，HTML (Hyper Text Transfer Protocol)，FTP (File Transfer Protocol)，POP3 (Post Office Protocol version3)，SMTP (Single Mail Transfer Protocol)などがある．

アプリケーションが TCP を利用してデータを送信するとき，データはプロトコル・スタックを下層に向かって転送され，最終的にビットの流れとしてネットワーク上を転送される．各層はデータが無事に相手先に到達できるように，各種のヘッダを追加していく．図 2 にこのプロセスを示す．

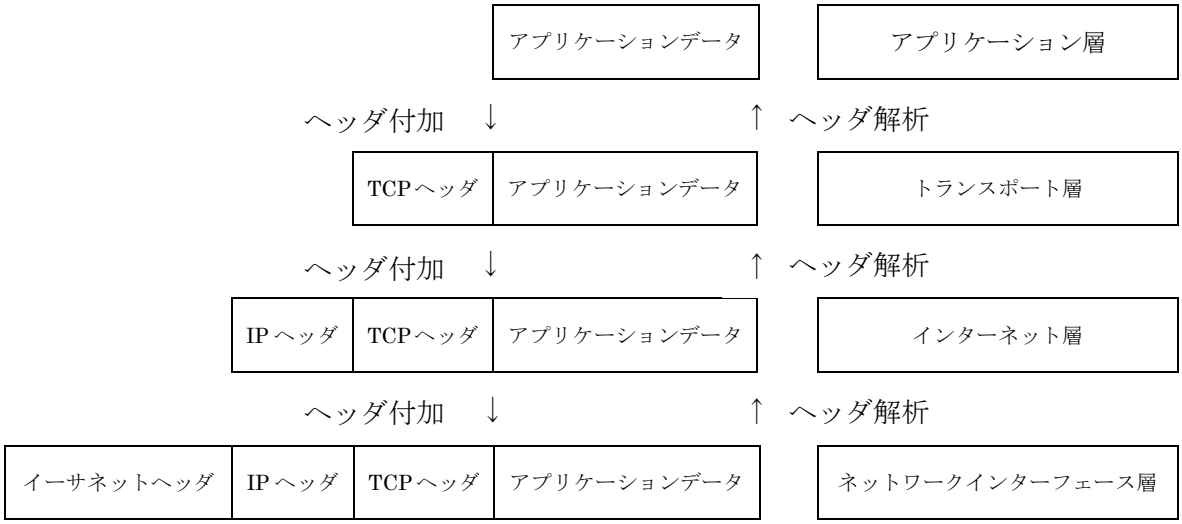

図 2．TCP/IP 階層によるデータ処理

2.2　TCP ヘッダ

図 3 のように，TCP データは IP データグラムにカプセル化される．

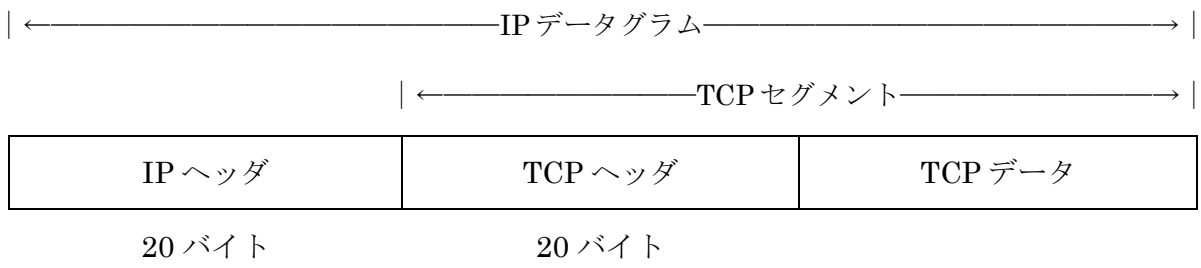

図 3．IP データグラムにカプセル化された TCP データ

さらに，図 4 で TCP ヘッダ [1]の形式を示す．

・送信元ポート番号

送信元のアプリケーションの使用するポート番号が挿入される．

・あて先ポート番号

送信先のアプリケーションの使用するポート番号が挿入される．

・シーケンス番号

送信するセグメントのデータ全体での位置を示す．データを送信する度に送信データのオクテット数だけ値が加算される．

・確認応答番号

次に受信すべきデータのシーケンス番号が挿入される．送信側は返された確認応答番号より前までは正常に受信されたと判断する．

・ヘッダ長

TCP ヘッダの長さを示す．

・予約ビット

将来の拡張のために約束されているフィールドである．

・フラグビット

—NS (Nonce Sum): 輻輳保護.

—CWR (Congestion Window Reduced): ECN により輻輳ウィンドウを小さくしたことを示す.

—ECE (ECN-Echo): 通信相手にネットワークが輻輳していることを伝える.

—URG (Urgent Flag): 緊急ポインタの値が有効であることを示す.

—ACK (Acknowledgement Flag): 確認応答番号が有効であることを示す.

—PSH (Push Flag): データの受信側に対して，可能な限り早急にデータをアプリケーションに渡すことを要求する．すなわち，送信データの終了または区切りであることを意味する.

—RST (Reset Flag): コネクションを強制的に開放することを要求する.

—SYN (Synchronized Flag): コネクションを確立するために使用されるセグメントであることを示す.

—FIN (Fin Flag):コネクションの開放に使用されるセグメントを示す．これ以上送信すべきデータがないことを意味する.

・ウィンドウサイズ

TCP のデータ転送では，受信側で用意されたバッファの空き具合に応じて，送信側にデータを許可するフロー制御を導入している. この受信側の空きバッファの大きさは TCP ヘッダのウィンドウサイズで伝えられる．この部分には，空きバッファの大きさ（受信可能なデータのサイズ）をバイト単位で規定される.

・チェックサム

TCP ヘッダとデータの全体に対して，伝送誤りの検出を行うためのパラメータである.

・緊急ポインタ

データ転送中の割り込みキーの押下など, 優先的に送信すべきデータに対して使用さ

れるものである.

・オプション

TCP の通信で機能を追加するために用いられる.

<table>
<tr><td colspan="3">送信元ポート番号（16 ビット）</td><td>あて先ポート番号（16 ビット）</td></tr>
<tr><td colspan="4">シーケンス番号(32 ビット)</td></tr>
<tr><td colspan="4">確認応答番号(32 ビット)</td></tr>
<tr><td>ヘッダ長
(4 ビット)</td><td>予約
(3 ビット)</td><td>フラグビット</td><td>ウィンドウサイズ（16 ビット）</td></tr>
<tr><td colspan="3">TCP チェックサム（16 ビット）</td><td>緊急ポインタ（16 ビット）</td></tr>
<tr><td colspan="4">オプション</td></tr>
<tr><td colspan="4">データ</td></tr>
</table>

図 4．TCP ヘッダ

TCP はアプリケーション層に対して全二重サービスを提供する．全二重通信とは，双方向で行う通信において同時に双方からデータを送受信できる通信方式である．したがって，コネクションの双方のエンドはそれぞれ各方向に流れるデータのシーケンス番号を管理しなければならない.

TCP セグメントのデータ部分は任意である．コネクションが確立されたとき，およびコネクションが終了されたとき，いくつかのオプションをもつ TCP ヘッダだけが含まれたセグメントが交換される．データを持たないヘッダは，その方向に送られるデータがない場合，受け取ったデータの確認応答に利用される．また，データを含まないセグメントを送ることができるときにタイムアウトを処理する場合もある.

2.3　コネクションの確立

TCP のコネクションは，通信する二つのノードの IP アドレス，それぞれのプロセスが使用するポート番号の四つ組で識別される．

図 5 にコネクションの確立手順を示す．コネクションの確立において，まずクライアント側のプロセスからオープンが要求される．このオープンはコネクションを能動的に確立するという意味でアクティブオープン(Active Open)と呼ばれる．一方，サーバ側ではコネクションの確立要求を待つパッシブオープン(Passive Open)が指示される．クライアントからのアクティブオープンに対して，SYN フラグが 1 のセグメント(SYN セグメント)が送出される．このセグメントにはシーケンス番号が設定されており，これは「クライアント側はこの値からシーケンス番号を開始する」ことを意味し，初期シーケンス番号と呼ばれる．サーバ側はこの SYN セグメントを受信すると，SYN フラグと ACK フラグを 1 にしたセグメント(SYN+ACK セグメント)を返送する．このセグメントは，サーバ側で使用するシーケンス番号の初期値を通知し，同時に確認応答番号も知らせている．SYN+ACK セグメントに対しクライアント側は，ACK フラグのみが 1 になったセグメント(ACK セグメント)を返送する．このセグメントの確認応答番号は，サーバー側の初期シーケンス番号に 1 を加えたものである．このようにコネクションの確立 SYN，SYN+ACK，ACK の三つが交換される．この手順を 3 ウェイハンドシェイクと呼ぶ．

図 6 にコネクションの解放手順を示す．解放はプロセスからのクローズの要求に対して FIN フラグが 1 となったセグメント(FIN セグメント)が送信され，相手がそれに対する ACK セグメントを送出するという形で行われる．まず，クライアント側が FIN セグメントを送信，続いてサーバ側がそれを確認し，次にサーバ側が FIN セグメントを送信，続いてクライアント側が確認する．FIN セグメントの確認のためにシーケンス番号が 1 だけ使用されるのは SYN セグメントと同様である．FIN セグメントはアプリケ

ーションからのクローズにより送信されるため，サーバ側でクライアントからの FIN セグメントを確認応答する ACK セグメントと，サーバ側からの解放を通知する FIN セグメントは別々のタイミングで送出される．クライアント側からの FIN セグメントを受信した後，サーバ側がさらにデータを送信し，その後 FIN セグメントを送信するという通信も可能となる．このような通信をハーフクローズ(half close)と呼ぶ． [2]

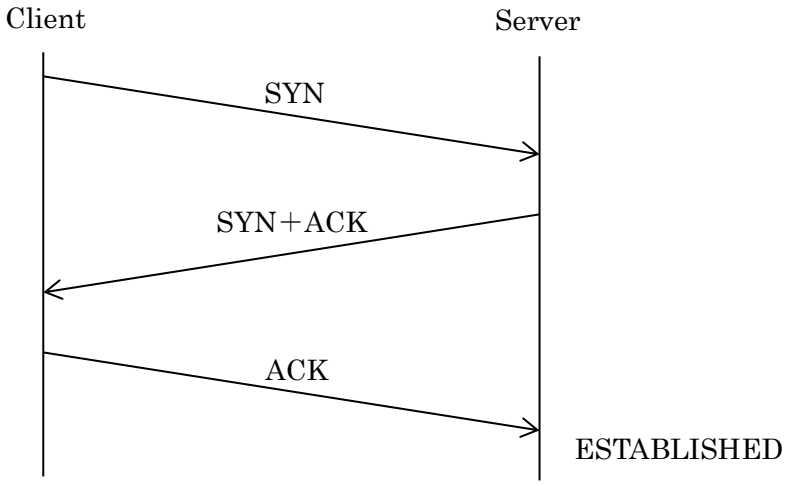

図 5．コネクションの確立

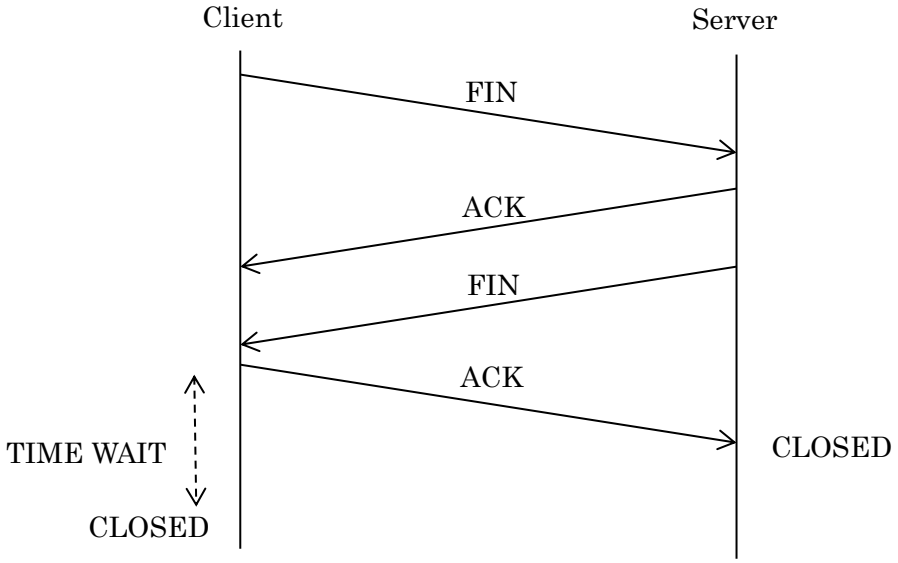

図 6．コネクションの解放

2.4　再送処理

データを転送している最中に，伝送エラーによりビット誤りが生ずる場合がある．このときは, データリンクプロトコルの FCS や TCP のチェックサムにより誤りが検出され，フレームまたはセグメントが破棄される．また IP では，ルータのバッファを超える IP データグラムが送られ，パケットが破棄される場合がある．このような状況に対して，TCP では紛失したデータセグメントを再送する二つの方式をサポートしている．

一つはタイムアウト再送 [2]である．TCP ではタイムアウト再送を行うために，データを転送している最中に相手との往復遅延時間(Round Trip Time)を決定する．初期の TCP では，以下のように決められていた．

$$R = \alpha R + (1 - \alpha)M$$

$$\mathrm{RTO} = R\beta$$

このとき，Mはデータセグメントを送信してから ACK セグメントを受信するまでの時間の測定値，Rは RTT の計算値，RTO はタイムアウト時間，α，βは定数である．

しかしこの式では, 遅延時間が急に増加した場合などに対応できないという指摘により，1988 年に以下のような計算式が導入された．

$$Err = M - A$$

$$A = (1 - g)A + gM$$

$$D = (1 - h)D + Mh|Err|$$

$$\mathrm{RTO} = A + 4D$$

このとき，Aは平均の RTT 値の計算値，Errは RTT の測定値からその時点での平均値を引いた値，DはErrの絶対値の平均であり RTT の平均偏差，g，hは平滑化のための定数である．この方式により，RTT の変動に対しても，より的確な RTO を求めることが可能となったと考えられる．このように，RTT を測定しながら，再送タイムアウト時間 RTO を決定する. RTO の時間が経過しても確認応答が取れないデータセグメント

を再送する．この方式を，タイムアウト再送と呼ぶ.

もう一つは高速再転送 [2]である．あるデータセグメントが紛失した場合，それに続くデータセグメントを要求する ACK セグメントを送信し続けることになるため，送信側は紛失したデータセグメントを再送する方法が考えられる．TCP では，確認応答番号もウィンドウサイズも同一であるような ACK セグメントを三つ受信すると，その ACK の確認応答番号に対応するデータセグメントを再送する手順を導入している．これを高速再転送と呼ぶ．図 7 に高速再転送によるデータセグメントの挙動を示す.

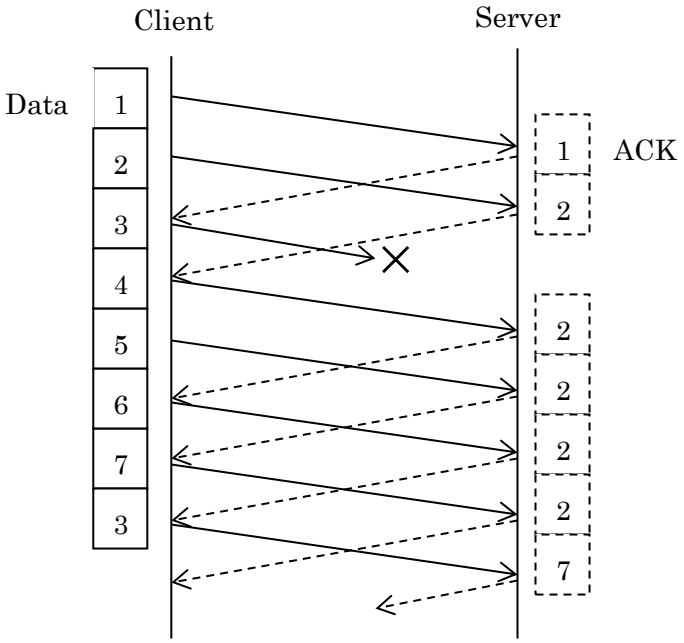

図 7．高速再転送によるデータセグメントの挙動

2.5　フロー制御

TCP では，データの受信側に用意された受信バッファの空き容量を，ウィンドウサイズとして送信側に通知し，送信側はその範囲でデータセグメントを送信する．この手順をフロー制御 [2]という．送信側がシーケンス番号のデータを送信できる範囲を「ウィンドウ」と呼び，さらにウィンドウの中で最も小さいシーケンス番号を lwe(lower window edge), 最も大きいシーケンス番号の次のシーケンス番号を uwe(upper window edge)と呼ぶ．なお，ウィンドウサイズを通知することを「広告 (advertise)」という．

この後，確認応答番号とウィンドウサイズを持つ ACK セグメントを受信すると，lwe=確認応答番号，uwe=確認応答番号＋ウィンドウサイズとしてウィンドウの変更が行われ, これに伴いウィンドウはスライドされる. このように TCP のデータ通信では，データを送信できるウィンドウが ACK の受信に伴い移動している．このようなフロー制御の方法をスライディングウィンドウプロトコルと呼ぶ.

2.6　輻輳制御

大量のパケットを受信すると混雑が生じ，パケットの到着に遅延が生じたり，またネットワークのキャパシティを超えるようなら超えた分のパケットは破棄されてしまう．このような現象を輻輳といい，TCP はこのような輻輳を回避するための制御を行う．

フロー制御では, 受信側から広告されたウィンドウサイズの範囲でデータセグメントを送信することを許可したが，これではネットワークに輻輳が発生した場合でも，それとは無関係にデータを送信し続けてしまう．そこで，データの送信側が輻輳制御のためのウィンドウを新たに設ける. このウィンドウをフロー制御のためのウィンドウと区別して輻輳ウィンドウ(congestion window)と呼ぶ．データの送信側は，受信側が広告したウィンドウと，輻輳ウィンドウの小さい値の範囲で，データの送信を行う． [2]

第 3 章　　TCP の輻輳制御方式

本章では，TCP の基本的な輻輳制御方法と，現在までに提案されている輻輳制御アルゴリズムに関する概要を述べる．

3.1　TCP の輻輳制御

TCP は「スロースタート(Slow Start)」と「輻輳回避(Congestion Avoidance)」の 2 つのフェーズによって輻輳制御を実現している．[2]

スロースタートフェーズでは，クライアント側はまずデータセグメントを一つ送る．もし輻輳がなければ，問題なくそのセグメントの確認応答が返される．その場合，送信可能な範囲を 1 データセグメントだけ増やして，二つのデータセグメントを送る．このように，コネクションを確立直後には，新しいデータセグメントに対する確認応答を受信する毎に，輻輳ウィンドウを 1MSS 増加するという方法を用いる．増加前の輻輳ウィンドウを$cwnd$，増加後を$cwnd_{new}$，最大セグメント長をMSSとすると，

$$cwnd_{new} = cwnd + MSS$$

と表すことが出来る．

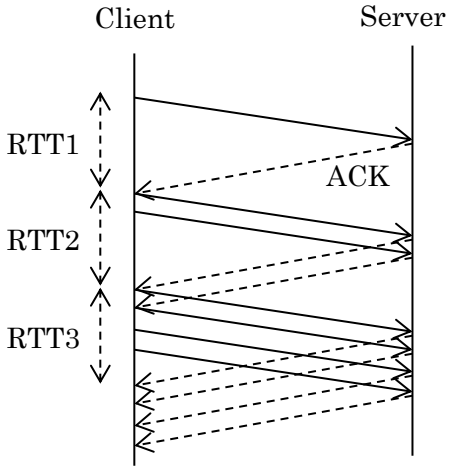

図 8．スロースタートフェーズ

輻輳回避フェーズでは，$cwnd$の増加においては，再送が行われた時点でのウィンドウサイズの半分の値まで$cwnd$が増加した後は，指数関数的ではなくリニアに増加させるようにするという方式を採用している．式で表すと

$$cwnd_{new} = cwnd + MSS \times \frac{MSS}{cwnd} = cwnd + \frac{MSS^2}{cwnd}$$

となる．これにより，つぎの輻輳が発生する時間を遅らせることが可能になり，より多くのデータを送信できるようになる．

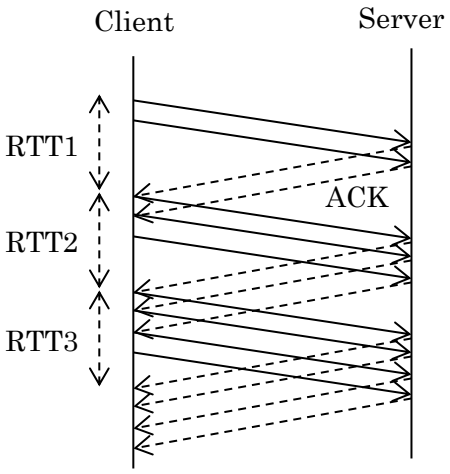

図 9．輻輳回避フェーズ

3.2　輻輳制御アルゴリズム

TCP は現在，様々なネットワーク状況に応じた様々な輻輳制御アルゴリズムが提案されている．大きく区別して初期の OS から搭載されている Standard TCP，近年の高速・広帯域ネットワークに対応した TCP Variants に分けられる．さらに TCP Variants には，パケットロスを輻輳の指標とする Loss-Based 方式，RTT の変化を輻輳の指標とする Delay-Based 方式，2 つを組み合わせた Hybrid 方式の 3 種類に分類される． [3]

以降で本研究に関連する TCP バージョンを解説する．

3.2.1 TCP Tahoe (Standard TCP)

TCP Tahoe [2]は，前述のスロースタートフェーズと輻輳回避フェーズによって構成されている．

まず TCP コネクションが確立されると，$cwnd$を 1MSS に，ssthresh を 65535 バイトに初期化する．コネクション確立時には ssthresh の値を大きくすることにより，輻輳回避フェーズへ移行することを防いでいる．輻輳が発生すると，その時点のウィンドウサイズの 1/2 を ssthresh に記憶する．ただし ssthresh の値は最小でも 2MSS とする．さらに$cwnd$の値を 1MSS とする．新しいデータに対する確認応答が送られてくると，$cwnd$を増加させる．このとき$cwnd$≦ssthresh なら，スロースタートフェーズを実行する．その場合$cwnd$は 1MSS 増加される．$cwnd$＞ssthresh なら，輻輳回避フェーズを実行する．その場合$cwnd$は$cwnd = cwnd + MSS \times \frac{MSS}{cwnd}$の式に基づき MSS を更新する．

図 10 に TCP Tahoe の輻輳ウィンドウの変化を示す．

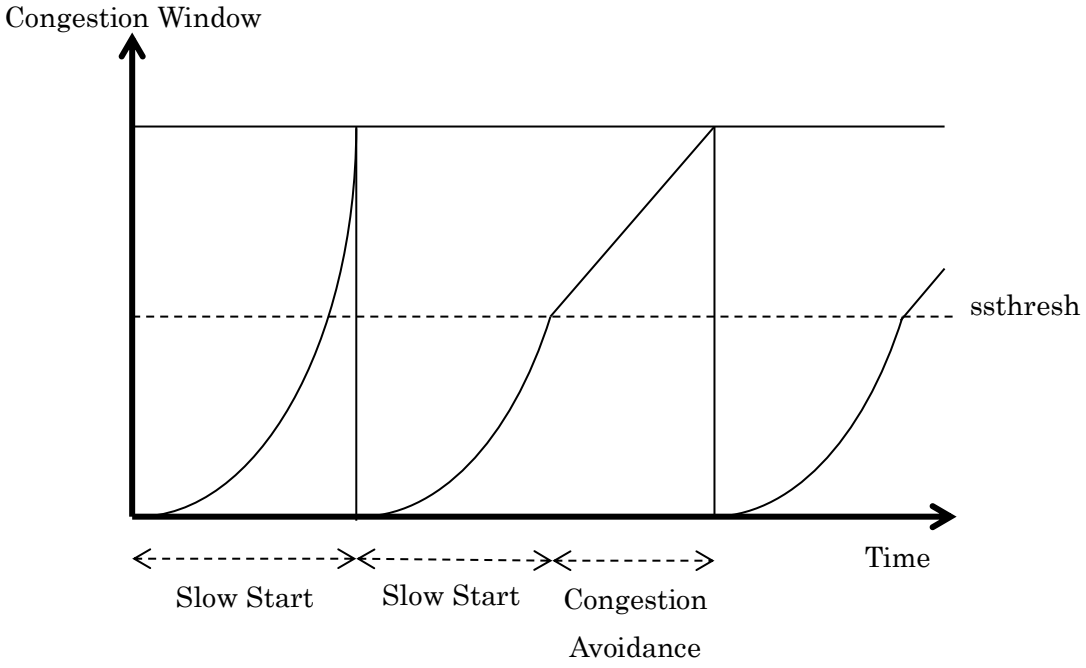

図 10．TCP Tahoe の輻輳ウィンドウの変化

3.2.2 TCP Reno (Standard TCP)

TCP Reno [2]は，前述した TCP Tahoe の機能に加えて輻輳回避フェーズに高速リカバリ(Fast Recovery)を採用したアルゴリズムである．

まず三つ目の重複 ACK を受信すると，その時点でのウィンドウサイズの 1/2 を ssthresh に記録する．ただし ssthresh の値は MSS の倍数に丸められ，1/2 とした結果が 2MSS より小さい場合は 2MSS の値とする．また，*cwnd*を ssthresh+3×MSS に設定し，紛失したデータセグメントを再送する．その後，別の重複 ACK を受信するごとに，*cwnd*を 1MSS だけ増加させ，新たな*cwnd*の値の範囲でデータセグメントを送信する．新しいデータセグメントを確認応答する ACK を受信すると，*cwnd*に ssthresh の値を設定する．その結果，*cwnd*は再送を行った時点のウィンドウサイズの 1/2 となる．このアルゴリズムにより，TCP Reno は輻輳ウィンドウを半分にしつつ，連続的なデータ転送を行うことができる．図 11 に TCP Reno の輻輳ウィンドウの変化を示す．

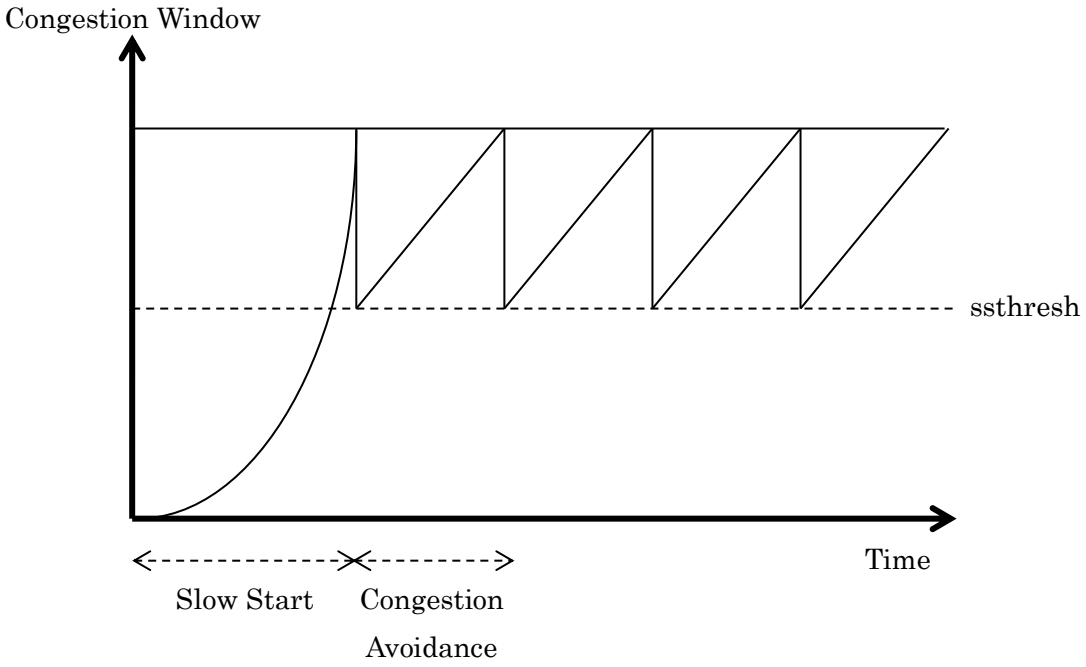

図 11．TCP Reno の輻輳ウィンドウの変化

3.2.3 BIC TCP (TCP Variants)

BIC TCP [4]は Loss Based 方式のプロトコルで，Adittive Increase モードと Binary Search モードの二つのウィンドウ増加フェーズを導入している．パケットロスが起きると，減少係数βによりウィンドウを減少させる．減少前にセットされたウィンドウサイズの最大値W_{max}と，減少前にセットされたウィンドウサイズの最小値W_{min}の中間点にジャンプすることにより Binary Search モードを行う．しかし，中間点にジャンプすることは 1RTT 内の増加が大きすぎることがある．そのため，中間点と現在の最小値の間の距離が一定値S_{max}よりも大きい場合，S_{max}により現在のウィンドウサイズを線形的に増加させる Additive Increase モードを用いる．輻輳ウィンドウが増加し前回のパケットロス直前の値を超えた場合，Binary Increase と Additive Increase モードと対称になるように輻輳ウィンドウを増加させる Max Probing モードになる．BIC TCP は前回のパケットロスが発生した時の輻輳ウィンドウサイズ付近に長く留まるため, 安定性がある．図 12 に BIC TCP の輻輳ウィンドウの変化を示す.

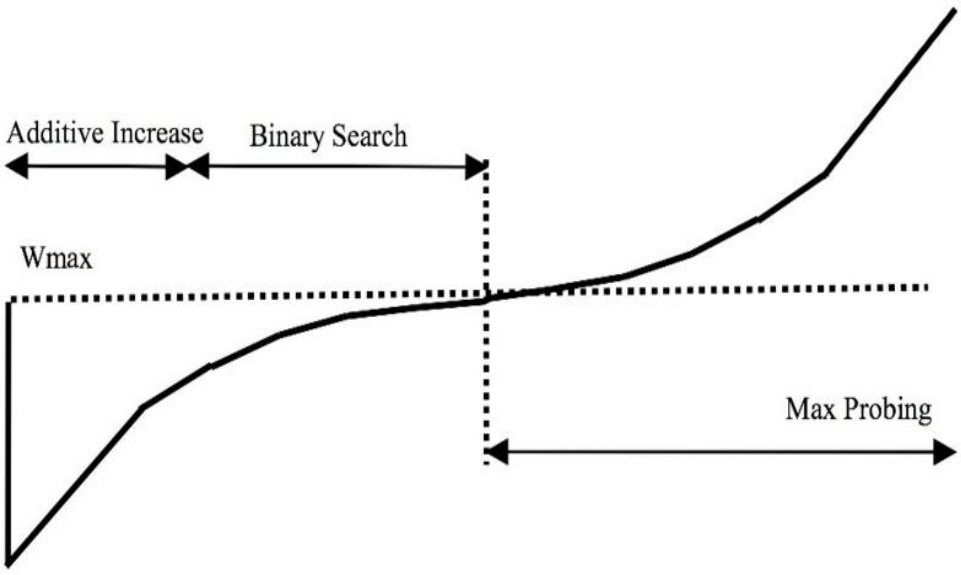

図 12．BIC TCP の輻輳ウィンドウの変化

3.2.4　CUBIC TCP(TCP Variants)

CUBIC TCP [4]は，BIC TCP の増加関数とよく似た形状をしている．CUBIC は簡易的にするように設計され，BIC のウィンドウ制御を高めている．

CUBIC の輻輳制御は次の関数によって決定されている．

$$\mathrm{W(t)} = C(t-K)^3 + W_{max} \qquad K = \sqrt[3]{\frac{W_{max}\beta}{C}}$$

Cは CUBIC の定数，βはパケットロス時の減少幅，W_{last_max}は前回のパケットロス時の輻輳ウィンドウサイズ，tは前回のパケットロスからの経過時間である．

ウィンドウはウィンドウの減少時に非常に早く増加し，しかしそれがW_{max}に近づくにつれて, その成長が遅くなる. CUBIC はウィンドウがはじめにゆっくりと成長する，より大きい帯域幅の探索を開始し，W_{max}から離れていくにつれてその成長を加速させている．このW_{max}周辺の遅い成長は，プロトコルの安定性を高め，W_{max}から離れることによる早い成長がプロトコルのスケーラビリティを確保している間, ネットワークの利用率を増加している．図 13 に CUBIC TCP の輻輳ウィンドウの変化を示す.

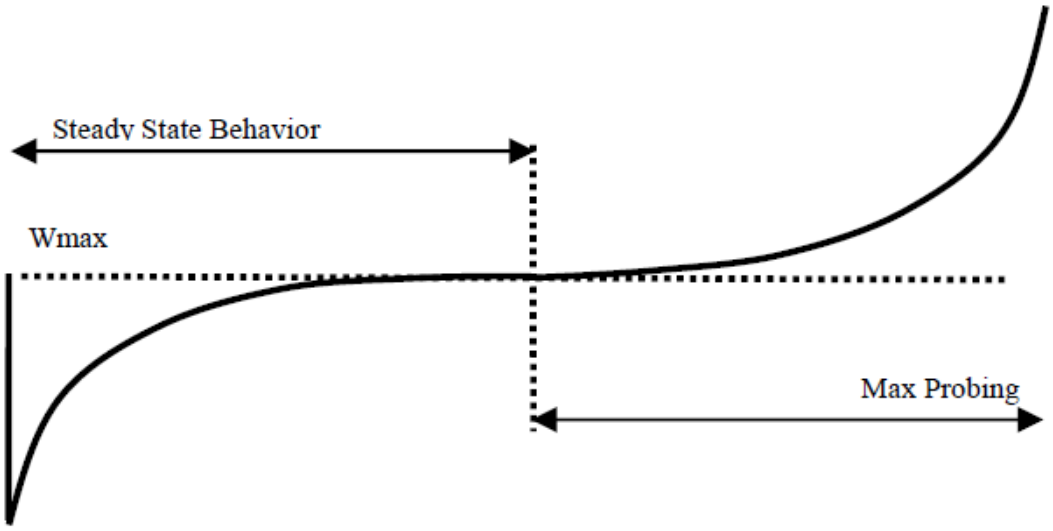

図 13．CUBIC TCP の輻輳ウィンドウの変化

3.2.5 TCP Vegas (TCP Variants)

TCP Vegas [5]は輻輳状態のネットワークを推定する方法として推定スループットと観測スループット間の違いを利用している．まず，Vegas は最小の測定 RTT である$BaseRTT$をセットし，推定スループットを以下の式で計算する．

$$expected = \frac{WindowSize}{BaseRTT}$$

このとき，WindowSize は現在のウィンドウサイズである．次に，Vegas は現在の実スループットを次のように計算する．各パケットが送信されると，Vegas はシステムクロックをチェックすることでパケットの送信時間を記録し，ACK が返る前に経過時間を計算することで RTT を計算する．その後，この推定 RTT を利用して，実スループットを次に従って計算する．

$$Actual = \frac{WindowSize}{RTT}$$

その後，Vegas は$expected$と$Actual$を比較し，差を計算する．

$$Diff = expected - Actual$$

$Diff$は正値であり，これはウィンドウサイズを調整するために使用される．さらに，任意の 2 つの閾値α，βを定義する．このとき，$0 \leq \alpha < \beta$である．もし$Diff < \alpha$なら，Vegas は次の RTT までの間ウィンドウサイズを線形に増加させる．もし$Diff > \beta$なら，Vegas は次の RTT までの間ウィンドウサイズを線形に減少させる．その他なら，ウィンドウサイズを変化させない．$Actual$が$expected$よりも遥かに小さい場合，それはネットワークが輻輳している可能性があることを示唆している．また，$Actual$が$expected$にとても近似している場合，コネクションは利用可能なフローレートを利用していないと判断してフローレートを増加させている．

3.2.6 TCP Veno (TCP Variants)

TCP Veno [6]はプロアクティブにウィンドウサイズを調整する方法ではなく，コネクションが輻輳状態であるかどうかを指標として N の測定値を用いる．ここで N は次の式で計算される．

$$\mathrm{N} = Actual \times (RTT - BaseRTT) = Diff * BaseRTT$$

Vegas におけるバッファ推定が大きい場合($\mathrm{N} > \beta$)は輻輳回避フェーズの間輻輳ウィンドウの増加を制限し，ファストリカバリ状態で輻輳が発生すると$cwnd$を半分に設定する．これは言い換えると，パケットロスを検出し，かつ$\mathrm{N} > \beta$である場合$cwnd$が半分となるので，パケットロスを検出しただけならば現在の輻輳ウィンドウの 80%までしか減少しないようになっている．TCP Veno は輻輳回避フェーズにより長く留まる傾向にあるが，これはネットワークリソースを発見するための追加遅延としての価値がある．

3.2.7 HS TCP (TCP Variants)

HS TCP(High Speed TCP) [7]は基本的には TCP Reno と同じアルゴリズムであるが，輻輳回避フェーズにおける 1RTT ごとのウィンドウの増加幅，重複 ACK の受信によってパケット廃棄を検出した際のウィンドウサイズの減少幅が異なる．HS TCP は，現在のウィンドウサイズがW_{low}より小さい場合，TCP Reno と同じアルゴリズムに従ってウィンドウサイズを増減させる．一方，現在のウィンドウサイズがW_{low}より大きい場合，ACK パケットを受信した際に TCP Reno のようにウィンドウサイズを減少させない．ウィンドウサイズの増加幅および減少幅は，現在のウィンドウサイズを用いて決定される．ウィンドウサイズをwとすると，1RTT ごとのウィンドウサイズの増加幅は$a(w)$，重複 ACK によってパケット廃棄を検出した際のウィンドウサイズの減少幅は$b(w)$ となり，次の式で表される．

$$a(w) = \frac{2w^2 \cdot b(w) \cdot p(w)}{2 - b(w)}$$

$$b(w) = \frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})}\left(b_{high} - 0.5\right) + 0.5$$

$$p(w) = exp\left[\frac{\log(w) - \log(W_{low})}{\log(W_{high}) - \log(W_{low})} \cdot \left\{\log\left(P_{high}\right) - \log(P_{low})\right\} + \log(P_{low})\right]$$

ここで，$P_{low} = \frac{1.5}{{W_{low}}^2}$であり，TCP Reno において平均ウィンドウサイズがW_{low}となるときのパケット廃棄率を示している．HS TCP は，ネットワークにおけるパケット廃棄率がP_{high}のときに平均ウィンドウサイズがW_{high}になるように，ウィンドウサイズの増加幅$a(w)$，減少幅$b(w)$を決定する．また，現在のウィンドウサイズがW_{high}のとき，重複 ACK の受信によってパケット廃棄を検出した際のウィンドウサイズを$\left(1 - b_{high}\right) \cdot W_{high}$とする．

3.2.8 Scalable TCP (TCP Variants)

Scalable TCP [8]は Loss Based 方式のプロトコルで，HS TCP と同様に Reno の Congestion Avoidance Phase での制御を変更したものであり，広帯域ネットワークでの TCP スループットの向上を目指している．Scalable TCP では，$cwnd$の増加，減少のアルゴリズムを以下のように変更している．

$cwnd \leq lwnd$のとき

$$cwnd = \begin{cases} cwnd + \dfrac{1}{cwnd} & (ACK\text{受信時}) \\ cwnd - \dfrac{1}{2} & (\text{セグメントロス発生時}) \end{cases}$$

$cwnd > lwnd$のとき

$$cwnd = \begin{cases} cwnd + 0.01 & (ACK\text{受信時}) \\ cwnd - 0.125 \times \text{cwnd} & (\text{セグメントロス発生時}) \end{cases}$$

このとき，$lwnd$は TCP Reno と Scalable TCP のアルゴリズムを使い分ける閾値である．Scalable TCP では$cwnd$が$lwnd$より大きくなると$cwnd$を 1 つ増加していくため，$cwnd$の増加が TCP Reno よりも早く，高いスループットを得られる．

3.2.9 TCP Illinois (TCP Variants)

TCP Illinois [9]は Hybrid 方式のプロトコルで，スロースタートフェーズでは Reno と同様に Loss Based 方式，輻輳回避フェーズではウィンドウサイズのペースをキューイング遅延に合わせる. TCP Illinois は ACK が来るたびに$cwnd$ を$\frac{\alpha}{cwnd}$だけ増加させ，パケットロスが起きると$\beta * cwnd$だけ減少させる．このとき，αとβは変数であり，平均キューイング遅延時間d_aを参考に以下の式で決定する.

$$\alpha = f_1(d_a) = \begin{cases} \alpha_{max} & d_a \le d_1 \\ \dfrac{K_1}{K_2 + d_a} & otherwise \end{cases}$$

$$\beta = f_2(d_a) = \begin{cases} \beta_{min} & d_a \le d_2 \\ K_3 + K_4 d_a & d_2 < d_a < d_3 \\ \beta_{max} & otherwise \end{cases}$$

このときのK_1からK_4までの値は以下の式で決定する.

$$K_1 = \frac{(d_m - d_1)\alpha_{min}\alpha_{max}}{\alpha_{max} - \alpha_{min}}, \quad K_2 = \frac{(d_m - d_1)\alpha_{min}}{\alpha_{max} - \alpha_{min}} - d_1,$$

$$K_3 = \frac{\beta_{min} d_3 - \beta_{max} d_2}{d_3 - d_2}, \quad K_4 = \frac{\beta_{max} - \beta_{min}}{d_3 - d_2}$$

第4章　　提案方式

本章では，本研究の核の部分である提案手法について述べる．

4.1　提案概要

TCP は輻輳ウィンドウの数値分だけ MSS 単位に分割されたデータを送信する．分割されたデータには送るデータの先頭バイト番号と後尾データ番号がシーケンス番号として記録され，その番号に 1 を足した番号を ACK 番号として ACK に記録して送り返す．この番号を利用して，輻輳ウィンドウの推定を行う．

4.2　輻輳ウィンドウの値の推定

輻輳ウィンドウの推定は，ACK 番号とシーケンス番号を利用して行う．

1. i番目の ACK を受信してから．$i+1$番目の ACK を受信するまでに送信したデータの最大シーケンス番号seq_iとi番目の ACK 番号ack_iを記録する．
2. そのときの輻輳ウィンドウ$cwnd_i$の仮推定値を

$$cwnd_i = (seq_i - ack_i)/MSS$$

とする．

3. これをi番目の ACK の RTT の間だけ監視し，その間に計測した$cwnd_i$の中で最も大きい$cwnd_i$を，RTT 測定開始時点での$cwnd_i$の推定値とする．
4. これを通信終了時まで繰り返す．

以上の方法によって輻輳ウィンドウの推定値を計算する．

4.3　TCP の輻輳制御アルゴリズムの推定

4.2 で推定した値を用いて，各 TCP バージョンの推定を行う．

具体的には，推定値間の$cwnd$の増加量$\Delta cwnd$を求め，その増加量と$cwnd$の推定値を$(cwnd,\ \Delta cwnd)$としてプロットし，散布図として表す．

その散布図と，各 TCP バージョンの輻輳制御アルゴリズムに基づく輻輳ウィンドウの変化から予測される散布図を比較して推定値として適切かどうかを検証する．

なお，この方式は TCP の増加アルゴリズムに着目したものであるため，輻輳時の減少に着目した推定については，ここでは考えない．

4.3.1　TCP Reno の推定

TCP Reno の輻輳ウィンドウはスロースタートフェーズでは$cwnd$の値は RTT の間に指数関数的に増加する．このとき，ACK は 2RTT のうち 1 つの場合もある．つまり，$\frac{1}{2}cwnd \leqq \Delta cwnd \leqq cwnd$と推測できる．輻輳回避フェーズでは$cwnd$ の値は RTT の間に 1 ずつ増加するので，$\Delta cwnd$は常に 1 である．よって，TCP Reno では図 14 のような散布図の線上あるいは範囲内に推定値が重なると考えられる．

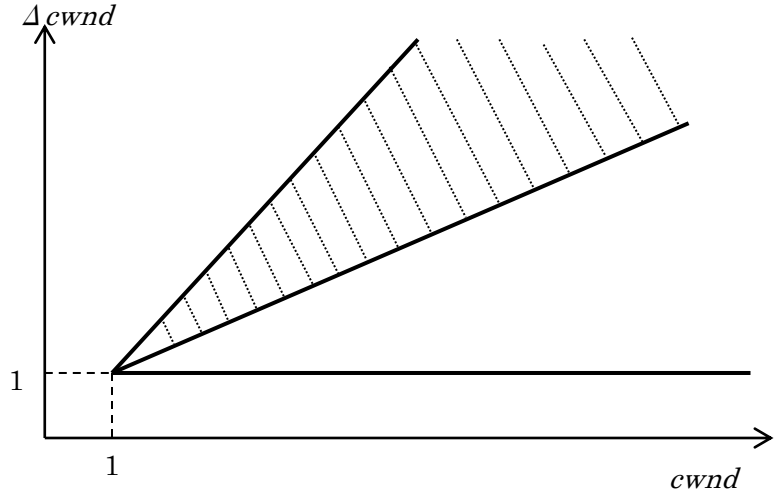

図 14．TCP Reno の散布予測図

4.3.2 TCP Vegas の推定

TCP Vegas の輻輳ウィンドウは，スロースタートフェーズでは RTT 単位で 2RTT に 1 回だけ$cwnd$を増加させるが，増加させる場合は TCP Reno と同様である．輻輳回避フェーズではボトルネックのキュー長Δを RTT に 1 回推定し，次のように$cwnd$を推定する．

$$cwnd = \begin{cases} cwnd + 1 & \text{if } diff < \frac{\alpha}{base\_rtt} \\ cwnd & \text{if } \frac{\alpha}{base\_rtt} < diff < \frac{\beta}{base\_rtt} \\ cwnd - 1 & \text{if } diff > \frac{\beta}{base\_rtt} \end{cases}$$

よって，TCP Vegas では図 15 のような散布図の線上あるいは範囲内に推定値が重なると考えられる．

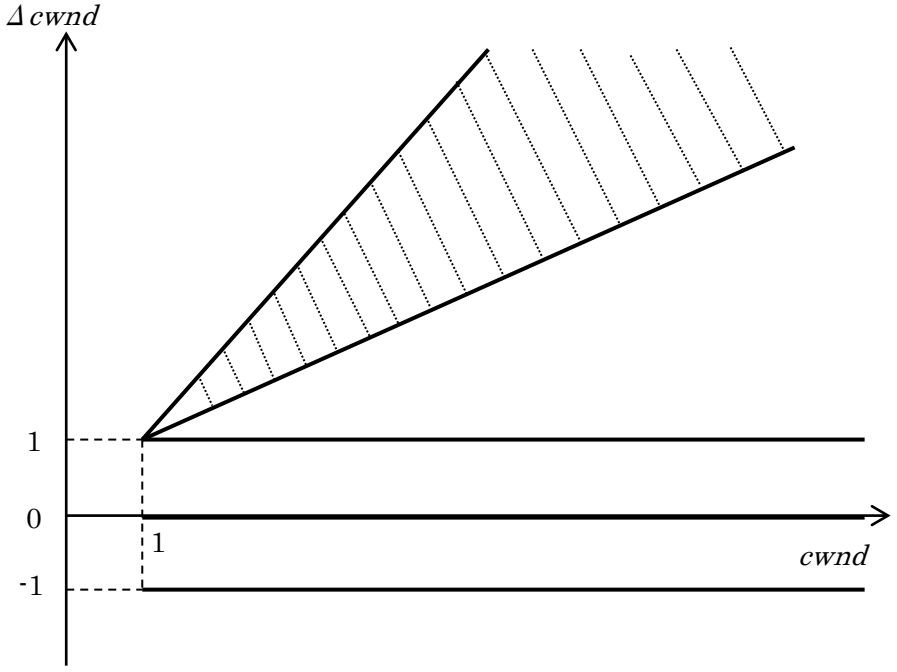

図 15．TCP Vegas の散布予想図

4.3.3 TCP Veno の推定

TCP Veno の輻輳ウィンドウは，スロースタートフェーズでは Vegas と同様，輻輳回避フェーズではボトルネックのキュー長Δを，Reno の輻輳制御の緩和に適用する．具体的にはΔ> βの場合には，2RTT に$cwnd$を 1 ずつ増加させる．この場合，RTT ごとにみると$cwnd$の増加量は 0 か 1 となる．よって，TCP Veno では図 16 のような散布図の線上あるいは範囲内に推定値が重なると考えられる．

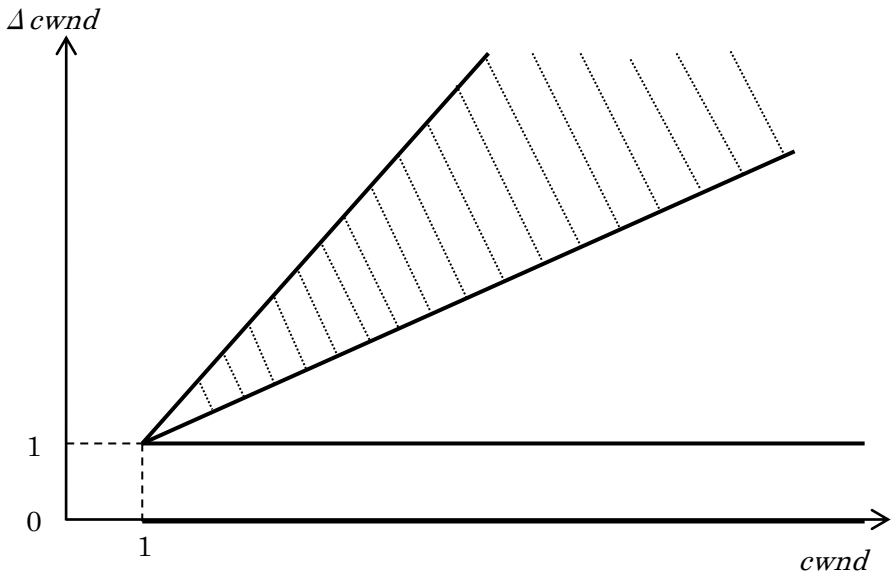

図 16．TCP Veno の散布予想図

4.3.4 CUBIC TCP の推定

CUBIC TCP では，$cwnd$を直前の輻輳現象からの経過時間の三次関数として与える．具体的には$cwnd$をw，経過時間をT，輻輳事象発生直前の$cwnd$をw_{max}とすると，

$$w = C\left(T - \sqrt[3]{\frac{W_{max}\beta}{C}}\right)^3 + W_{max}$$

で与えられる．したがって，RTT 間の$cwnd$の変化は

$$RTT \cdot \frac{dw}{dT} = RTT \cdot 3C(T - \sqrt[3]{\frac{W_{max}\beta}{C}})^2$$

さらにwの定義式から，$(T - \sqrt[3]{\frac{W_{max}\beta}{C}})^2$を$w$で表すと

$$RTT \cdot \frac{dw}{dT} = RTT \cdot 3\sqrt[3]{C}(w - w_{max})^{\frac{2}{3}}$$

となる．よって，RTT が$cwnd$に対して一定であるとすると CUBIC TCP では図 17 のような散布図の線上に推定値が重なるとかんがえられる．

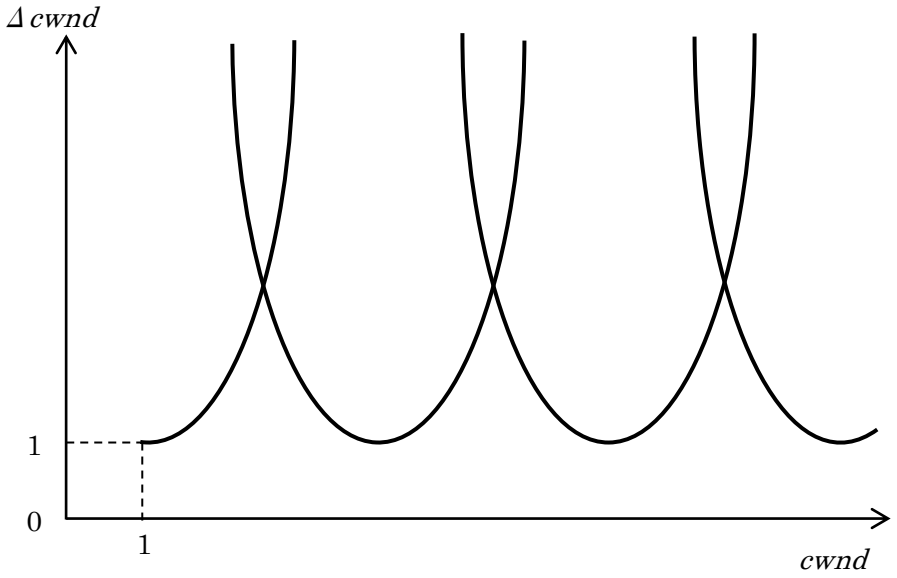

図 17．CUBIC TCP の散布予想図

4.3.5 TCP Illinois の推定

TCP Illinois の輻輳ウィンドウは，スロースタートでは Reno と同様，輻輳回避フェーズではネットワークキューイング遅延に従い，RTT 期間の*cwnd*の増加値を 0.1 から 10 まで変化させる．よって，TCP Illinois では図 18 のような散布図の線上あるいは範囲内に推定値が重なると考えられる．

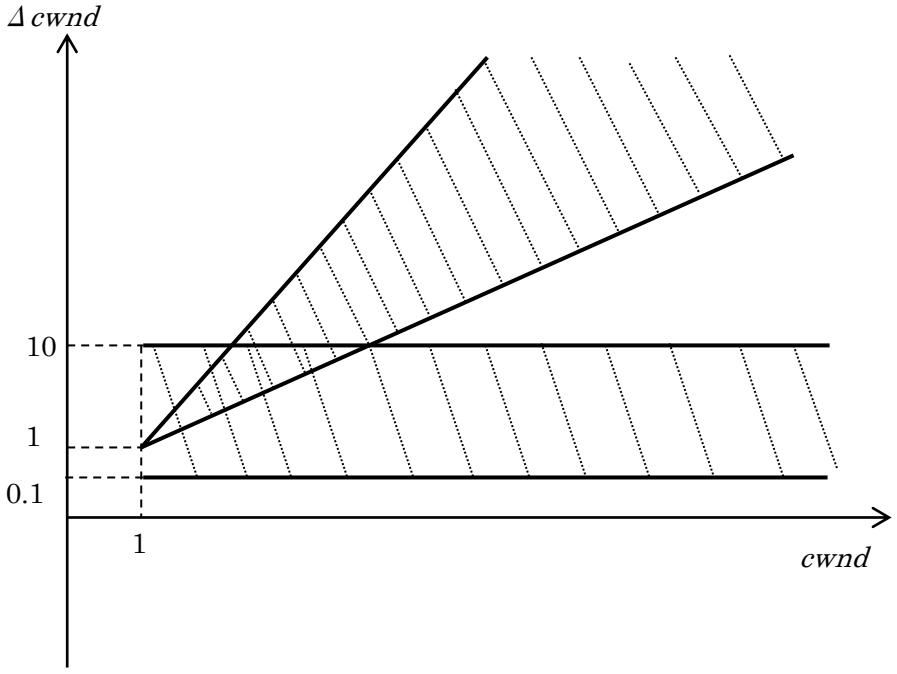

図 18．TCP Illinois の散布予想図

4.4　関連研究との比較

TCP に対して通信ログを用いて使用されているバージョンを推定しようという研究は各方面で行われている．古くは[10]が有名であり，通信ログのセグメントの送受信状態から，データの送信と ACK の受信，データの再送などを 1 つずつ順に識別し，それに基づいて TCP の内部状態を推測し cwnd の時間的変化を推定しようとするものである．

[11]も同様な手法を採用しており，これを用いて TCP のバージョンを推定することを目的としている．このために，複数の TCP バージョンに沿った TCP の内部状態を想定し，通信ログの結果から最もふさわしいバージョンを決めるという方法を用いている．しかしこれでは，Tahoe, Reno, NewReno の 3 つのバージョンのみを対象としている．

これに対して，[12]では通信ログから cwnd の時間的変化を推定し，その結果から，続いた ACK の受信による cwnd の変化の様子から，

- cwnd の変化が 1 増加であるものの割合
- cwnd の増加の速度
- cwnd が変化しないものの割合
- cwnd の変化の割合自体が変化した割合

の 4 つの特徴量を抽出する．さらに現在 Linux 上で実装されている 13 の TCP バージョンを対象として，そのうち任意の 2 つが混在している通信ログから，4 つの特徴量を用いて，その 2 つを区別可能であるかどうかを調べている．すなわち，この研究ではどの 2 つの TCP バージョンが用いられているかが既知であるという前提に立って，フローの TCP バージョンを区別することを目指している．

[13]では，TCP のバージョンをリアルタイムに区別することを目的としている．手法は[10]や[11]と同様に，データや ACK の送受信の状況から TCP の内部状態を推定することに基づいている．しかし，アルゴリズムは TCP が NewReno に基づいて動作して

いることを前提にしていると思われ，NewReno に従った TCP が正しく動いているかどうかをリアルタイムで判断する手法ととらえることができる．

[14]は，[10]から[13]および本論文の方法と異なり，アクティブな手法に基づいている．すなわち，通信をモニタリングした結果の通信ログに基づいて TCP バージョンを推定するのではなく，テストシステムが推定対象とするシステムと，意図した通信シーケンスで通信を行い，その結果からバージョンを推定する．対象は Linux に実装された 13 種類のバージョンである．具体的には，以下の手法を用いる．

- テストシステムが推定対象のシステムにデータを送信させる．
- テストシステムがデータに ACK を返すタイミングを調整することにより，1 秒または 0.8 秒の RTT を使用した 2 つのパターンを使用する．
- TCP コネクションの開始からスロースタートを行わせ，10RTT においてパケットロスを発生させ（データを受信しても ACK を返さない），タイムアウト再送を行わせる．
- その後は，すべてのデータに ACK を送信し，スロースタートと輻輳回避を行わせる．
- これらの動作を行いつつ，推定対象システムから送信されるデータを数え，cwnd の時間的変化を推定する．
- その結果から，cwnd の減少パラメータと，5 次式で近似した cwnd の増加関数を推定し，それに基づき TCP のバージョンを推定する．

これに対して，本論文で提案する手法は，以下のような特徴を持つ．

1. CUBIC や HS TCP などの新しい TCP のバージョンを含めて，通信対象の TCP について前提となる知識を必要としない．
2. cwnd とその変化の割合の対応だけを考えるため，[9]，[10]，[12]のように TCP の内部状態を推定する必要はない．このため，TCP 通信の途中からの通信ログでも

対応可能である．

3. 提案方式は現在のところ，cwnd の増加にのみ着目している．再送時の cwnd の減少に着目して，より推定の精度を上げることができる．

第５章　　エミュレーションによる実験結果

本章では，提案手法に基づいて行われたエミュレーションによる実験の詳細を述べ，結果について考察する．

5.1　実験環境

図 19 のように，2 台の端末をそれぞれクライアントとサーバーとして通信を行う．また，クライアント・サーバー間に 1 台のブリッジを配置し，そのブリッジで遅延を発生させることで輻輳ウィンドウの挙動を確認する．

ネットワークの監視にはネットワーク測定ツール Wireshark [15]，Linux のカーネルモジュール tcpprobe [16]を用い, Wireshark で推定値の算出に使用する ACK 番号とシーケンス番号を，tcpprobe で実測された輻輳ウィンドウをモニタリングする．

通信の遅延は 0.5 秒と 1 秒の 2 パターン行う．

今回の実験で通信を用いた TCP は Reno，Vegas，Veno，CUBIC，Illinois の 5 種類である．

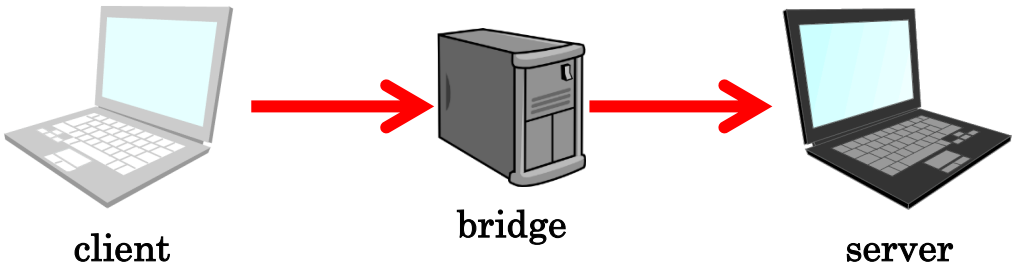

図 19．エミュレーション環境

5.2　実験結果

本研究では,

1.wireshark により観測された数値から推定された輻輳ウィンドウの推定値と tcpprobe により観測された輻輳ウィンドウの実測値の比較による推定値の有用性の検証

2.輻輳ウィンドウの推定値と TCP バージョンの輻輳制御アルゴリズムによる輻輳ウィンドウの散布予測図の比較による TCP バージョンの推定に使用可能かどうかの検証

の 2 つを行う.

なお，RTT 毎の輻輳ウィンドウの変化の散布図において，□のプロットが実測値,◇のプロットが予測値である.

5.2.1　TCP Reno の比較

図 20，図 21 に TCP Reno の RTT ごとの輻輳ウィンドウの変化を，図 22,図 23 に TCP Reno の輻輳ウィンドウと変化量の推移をそれぞれ遅延時間別に示す.

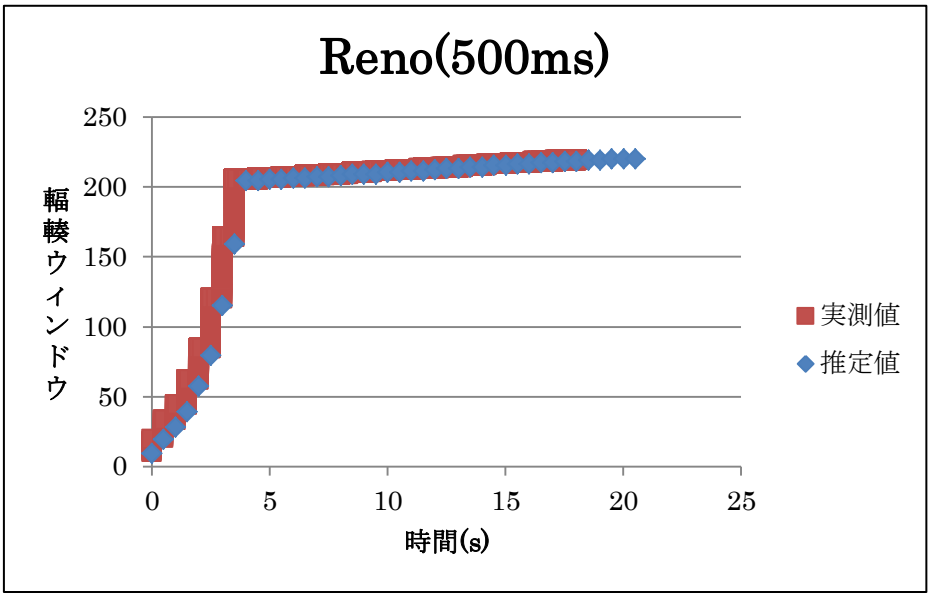

図 20. 500ms 遅延させたときの TCP Reno の RTT 毎の輻輳ウィンドウの変化

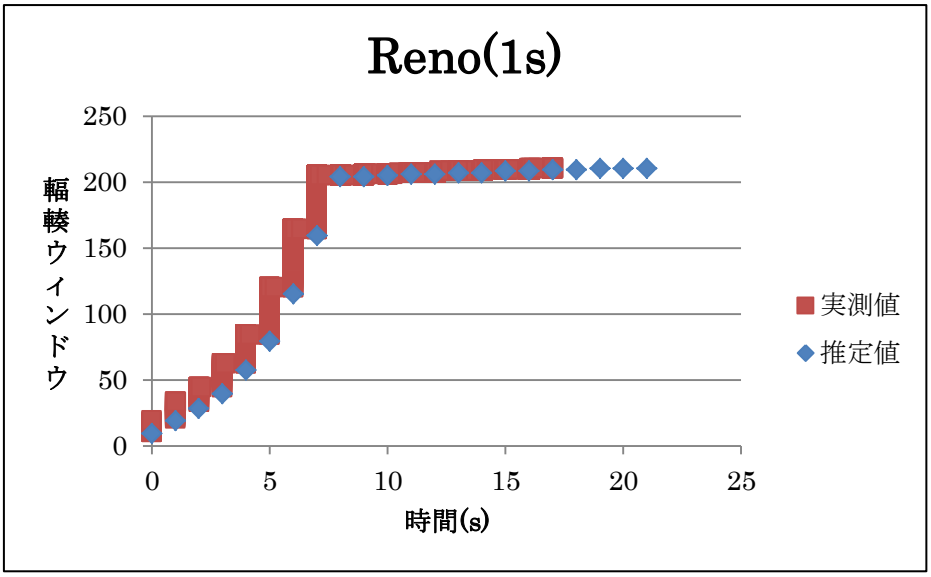

図 21. 1s 遅延させたときの TCP Reno の RTT 毎の輻輳ウィンドウの変化

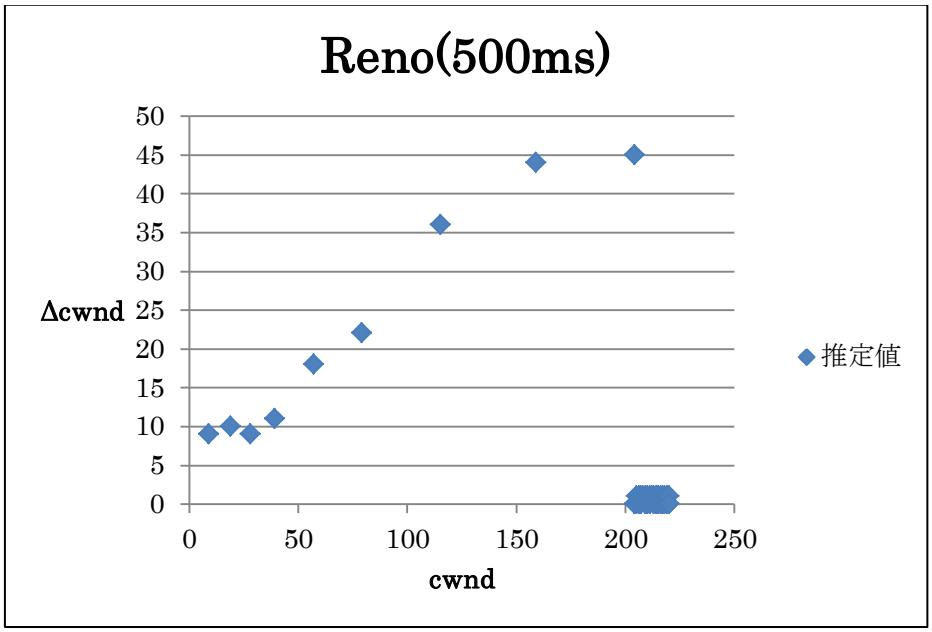

図 22. 500ms 遅延させたときの TCP Reno の輻輳ウィンドウと変化量の推移

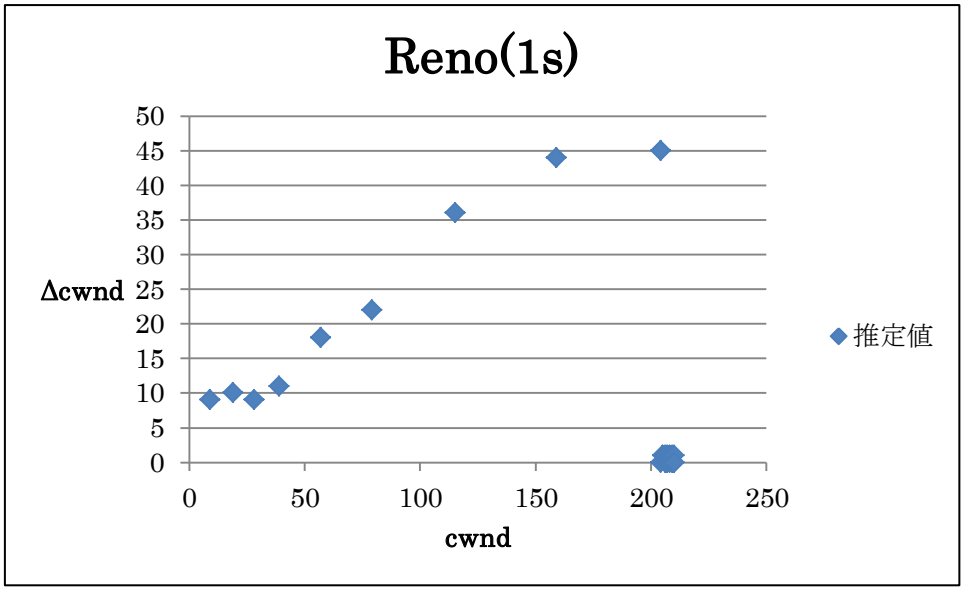

図 23. 1s 遅延させたときの TCP Reno の輻輳ウィンドウと変化量の推移

図 20，図 21 より，TCP Reno の推定値は実測値に非常に近似しているため，推定値としての有用性は高いと考えられる．

また, 図 22, 図 23 の TCP Reno のプロット推移は図 14 の散布予想図と比較すると，スロースタートフェーズにおいては非常に近似しているといえるが, 輻輳回避フェーズでは$\Delta cwnd$の値を見ると 1 だけでなく 0 が混在しているため, 有用であると断定できない．

5.2.2 TCP Vegas の比較

図 24，図 25 に TCP Vegas の RTT ごとの輻輳ウィンドウの変化を，図 26，図 27 に TCP Vegas の輻輳ウィンドウと変化量の推移をそれぞれ遅延時間別に示す．

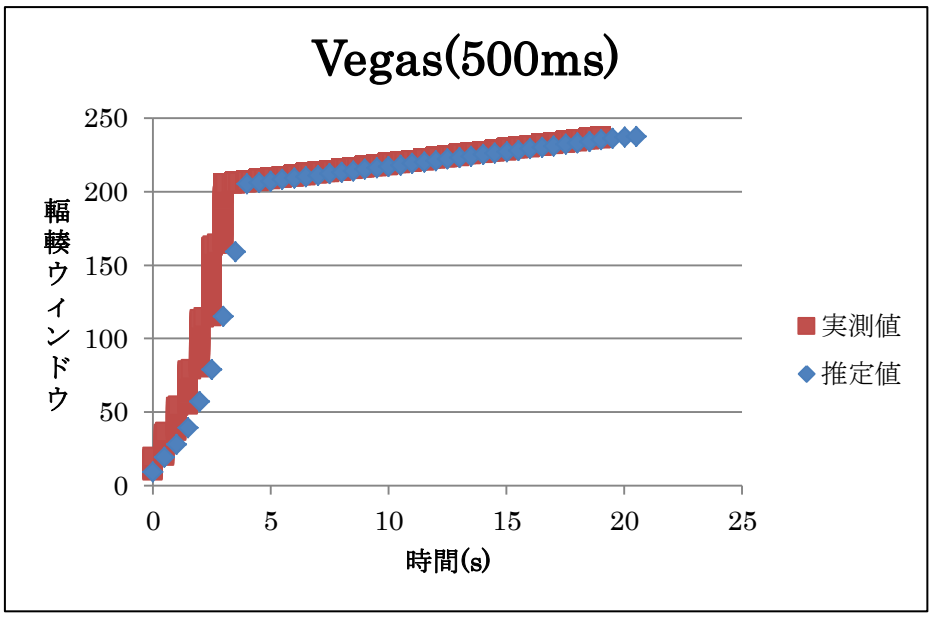

図 24．500ms 遅延させたときの TCP Vegas の RTT 毎の輻輳ウィンドウの変化

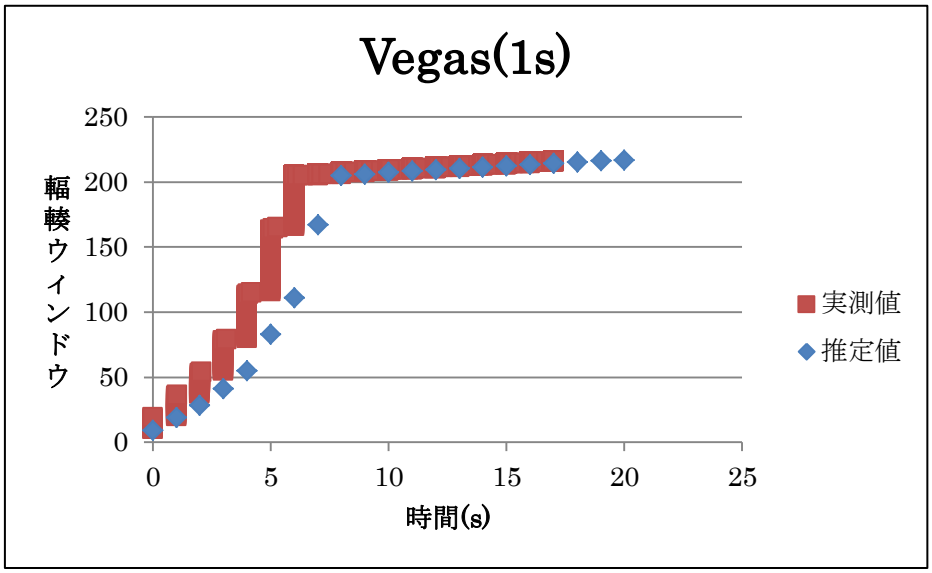

図 25. 1s 遅延させたときの TCP Vegas の RTT 毎の輻輳ウィンドウの変化

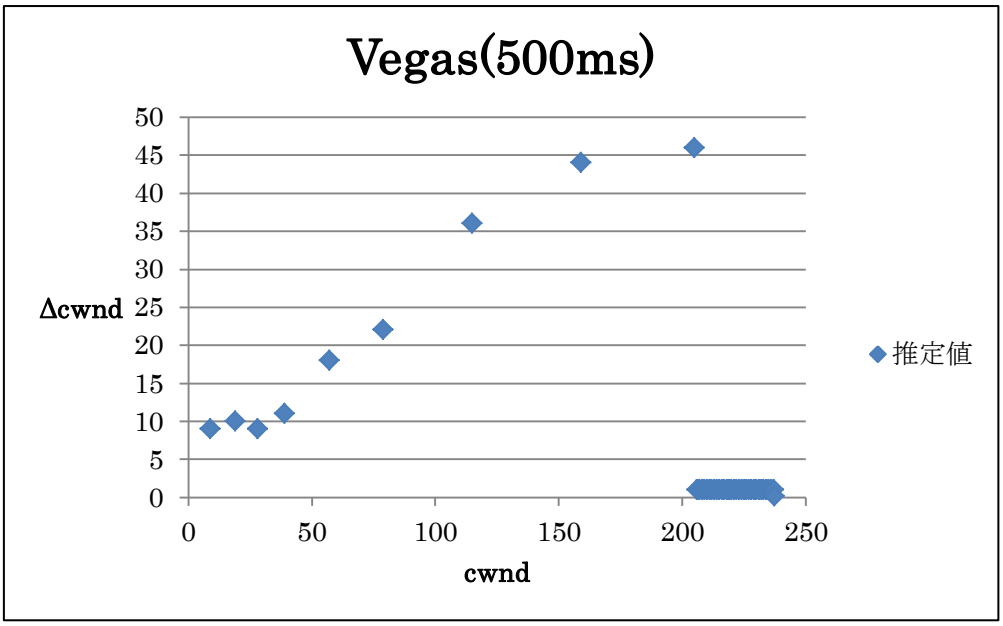

図 26. 500ms 遅延させたときの TCP Vegas の輻輳ウィンドウと変化量の推移

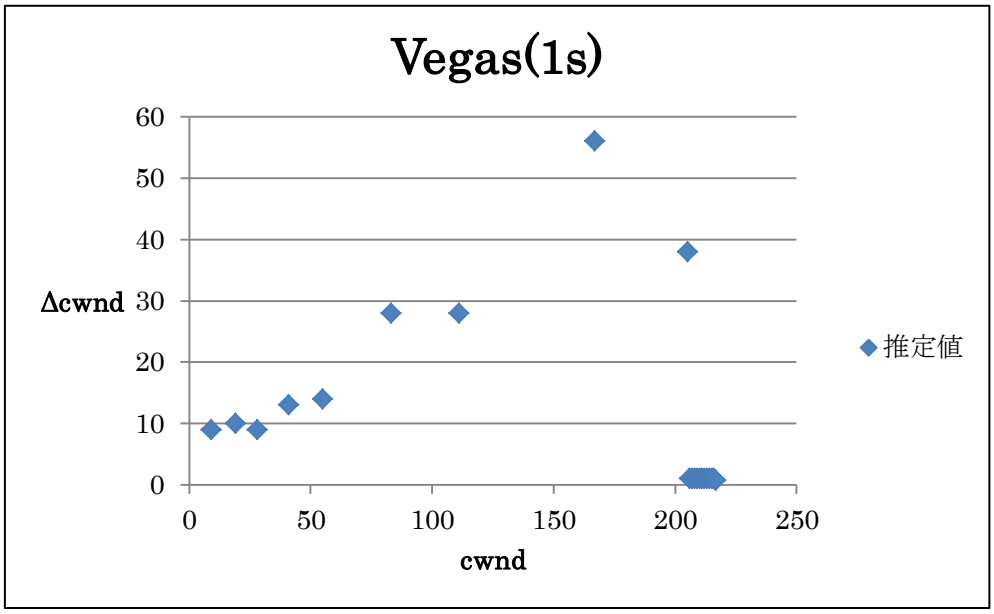

図 27. 1s 遅延させたときの TCP Vegas の輻輳ウィンドウと変化量の推移

図 24，図 25 より，TCP Vegas の推定値は実測値に非常に近似しているため，推定値としての有用性は高いと考えられる.

また, 図 26, 図 27 の TCP Vegas のプロット推移は図 15 の散布予想図と比較すると，スロースタートフェーズにおいては非常に近似しているといえるが, 輻輳回避フェーズでは$\Delta cwnd$の値を見ると 0 と 1 の値のみで-1 の値をとっていないため，有用であると断定できない.

5.2.3 TCP Veno の比較

図 28，図 29 に TCP Veno の RTT ごとの輻輳ウィンドウの変化を，図 30，図 31 に TCP Veno の輻輳ウィンドウと変化量の推移をそれぞれ遅延時間別に示す.

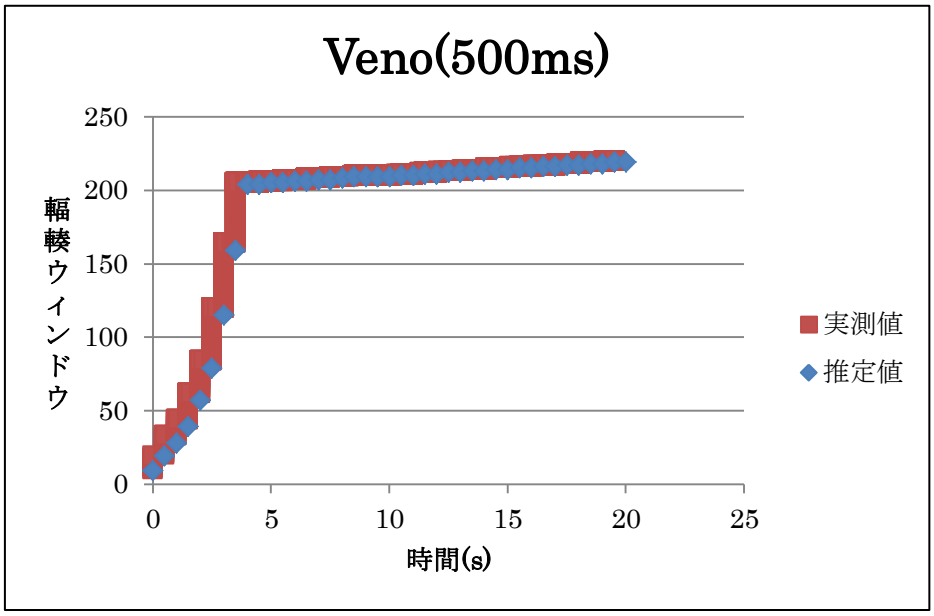

図 28.　500ms 遅延させたときの TCP Veno の RTT 毎の輻輳ウィンドウの変化

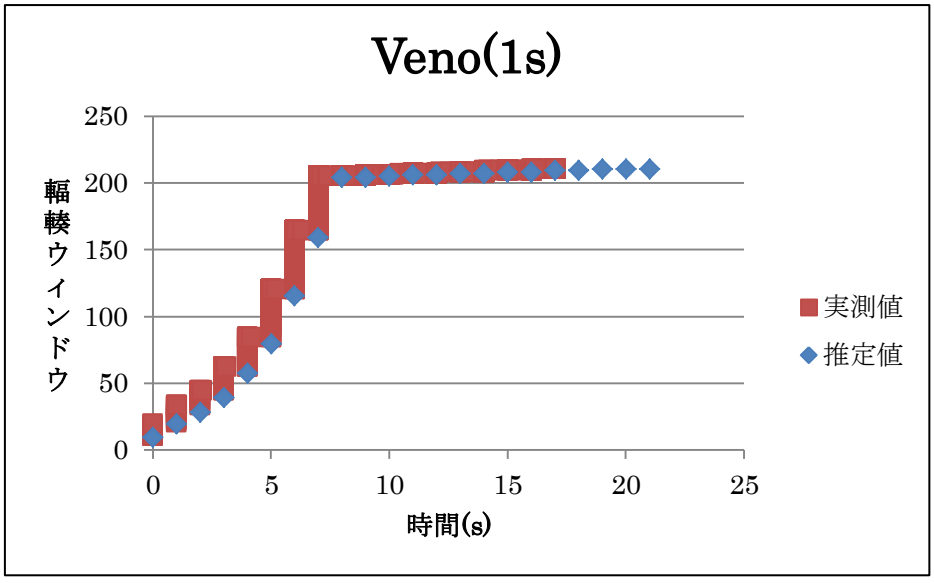

図 29.　1s 遅延させたときの TCP Veno の RTT 毎の輻輳ウィンドウの変化

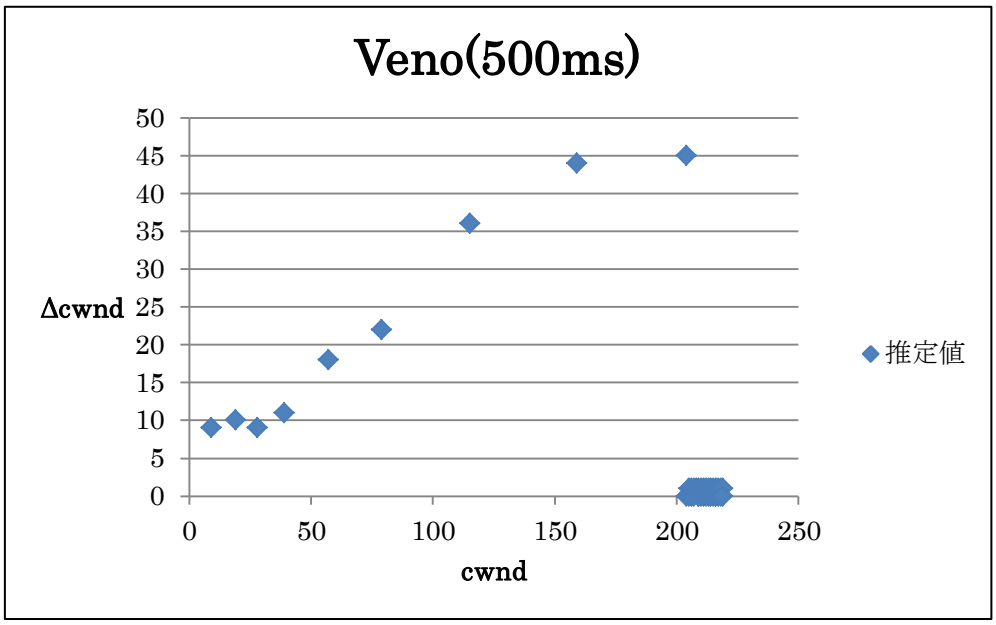

図 30．500ms 遅延させたときの TCP Veno の輻輳ウィンドウと変化量の推移

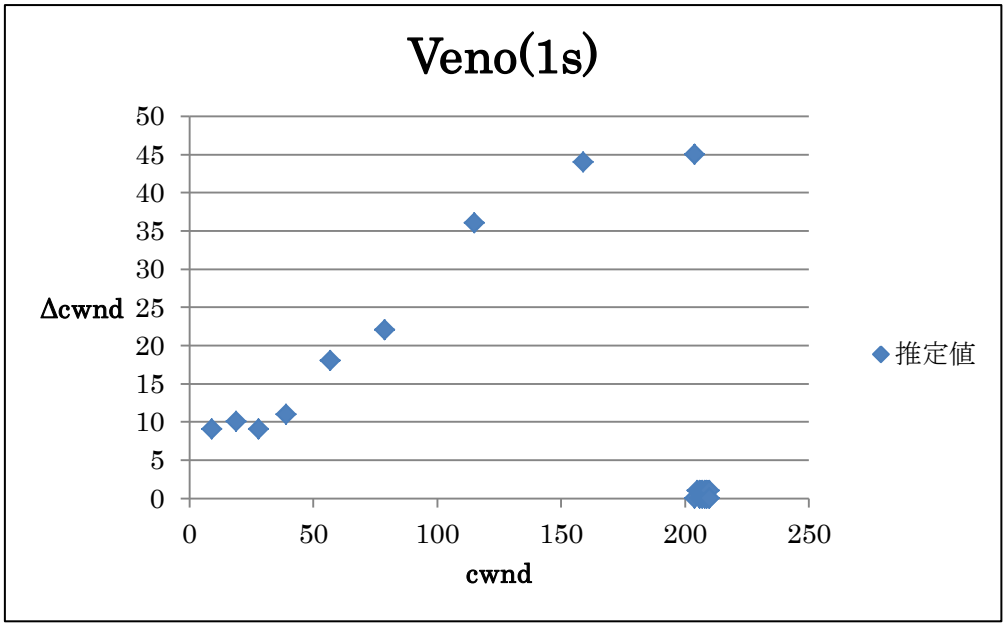

図 31．1s 遅延させたときの TCP Veno の輻輳ウィンドウと変化量の推移

図 28，図 29 より，TCP Veno の推定値は実測値に非常に近似しているため，推定値としての有用性は高いと考えられる．

また, 図 30, 図 31 の TCP Veno のプロット推移は図 16 の散布予想図と比較すると，

スロースタートフェーズにおいても輻輳回避フェーズにおいても非常に近似しているといえるため，有用性は高いと考えられる．

5.2.4　CUBIC TCP の比較

図 32，図 33 に CUBIC TCP の RTT ごとの輻輳ウィンドウの変化を，図 34，図 35 に CUBIC TCP の輻輳ウィンドウと変化量の推移をそれぞれ遅延時間別に示す．

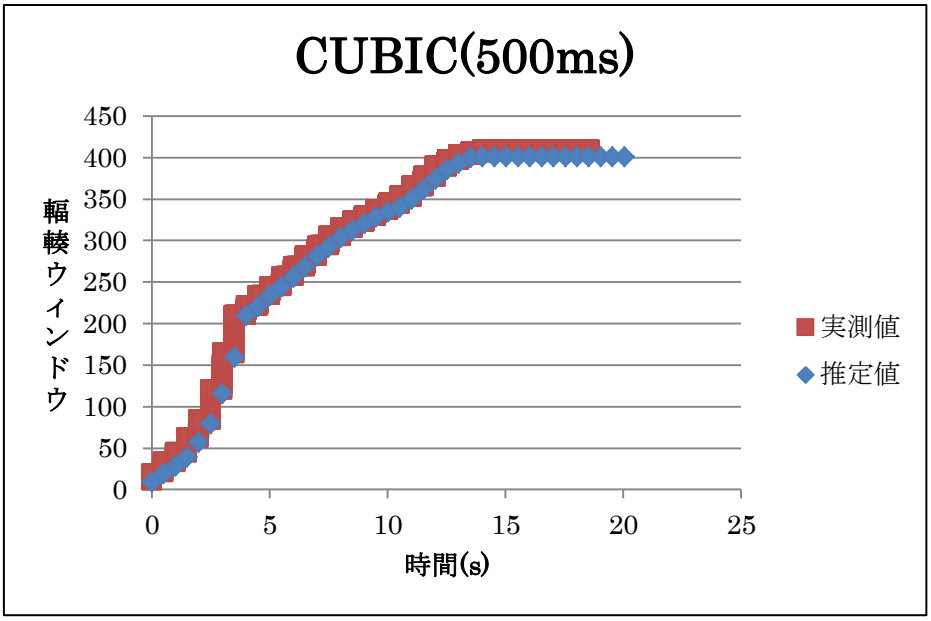

図 32.　500ms 遅延させたときの CUBIC TCP の RTT 毎の輻輳ウィンドウの変化

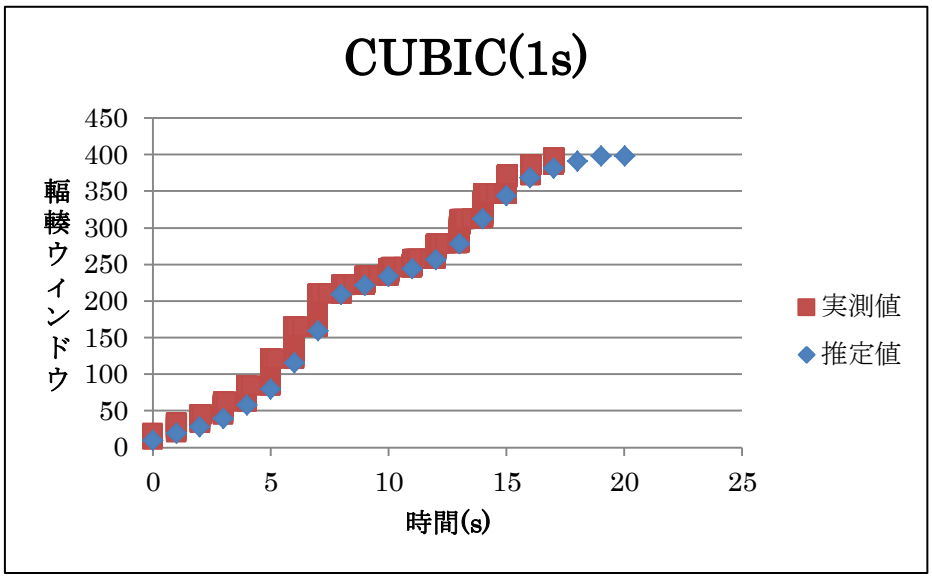

図 33.　1s 遅延させたときの CUBIC TCP の RTT 毎の輻輳ウィンドウの変化

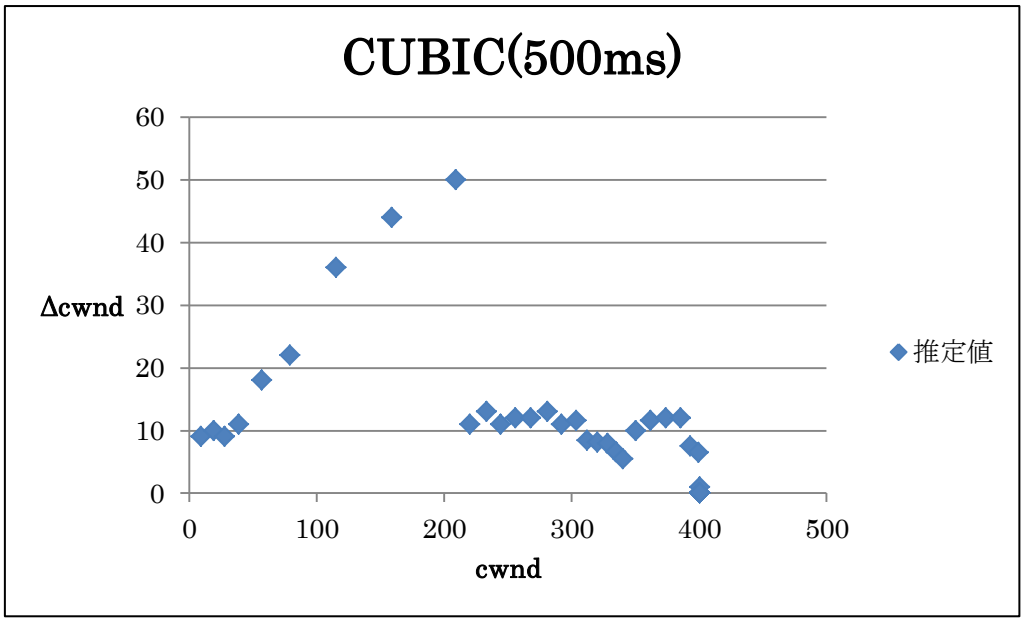

図 34.　500ms 遅延させたときの CUBIC TCP の輻輳ウィンドウと変化量の推移

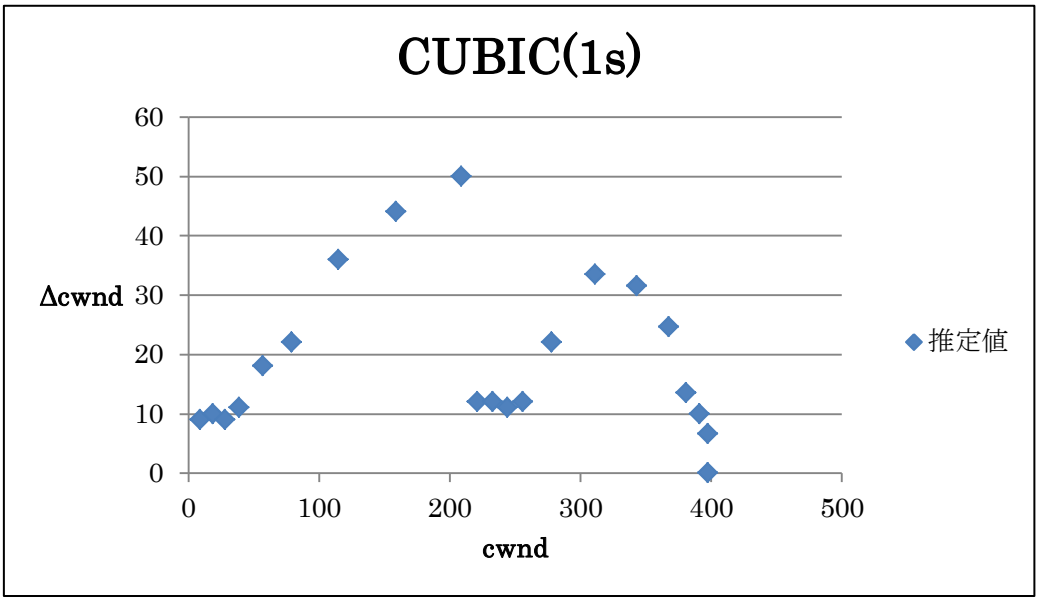

図 35．1s 遅延させたときの CUBIC TCP の輻輳ウィンドウと変化量の推移

図 32，図 33 より，CUBIC TCP の推定値は実測値に非常に近似しているため，推定値としての有用性は高いと考えられる．

また，図 34，図 35 の CUBIC TCP のプロット推移は図 17 の散布予想図と比較すると，スロースタートフェーズにおいては近似しているといえる．輻輳回避フェーズにおいては，図 36 の実線のような軌跡に非常に近似しているといえるため，有用性は高いと考えられる．

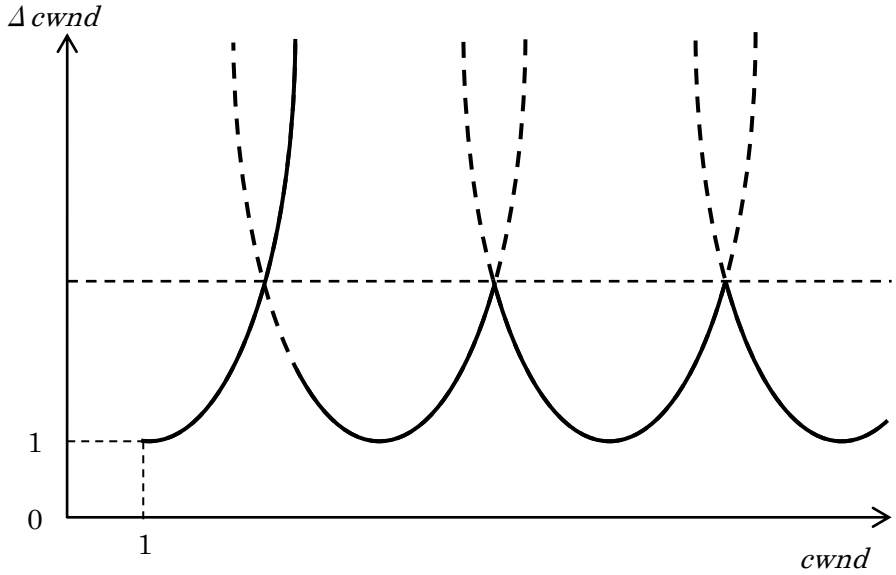

図 36. CUBIC TCP の散布予想図の詳解

5.2.5 TCP Illinois の比較

図 37，図 38 に TCP Illinois の RTT ごとの輻輳ウィンドウの変化を，図 39，図 40 に TCP Illinois の輻輳ウィンドウと変化量の推移をそれぞれ遅延時間別に示す.

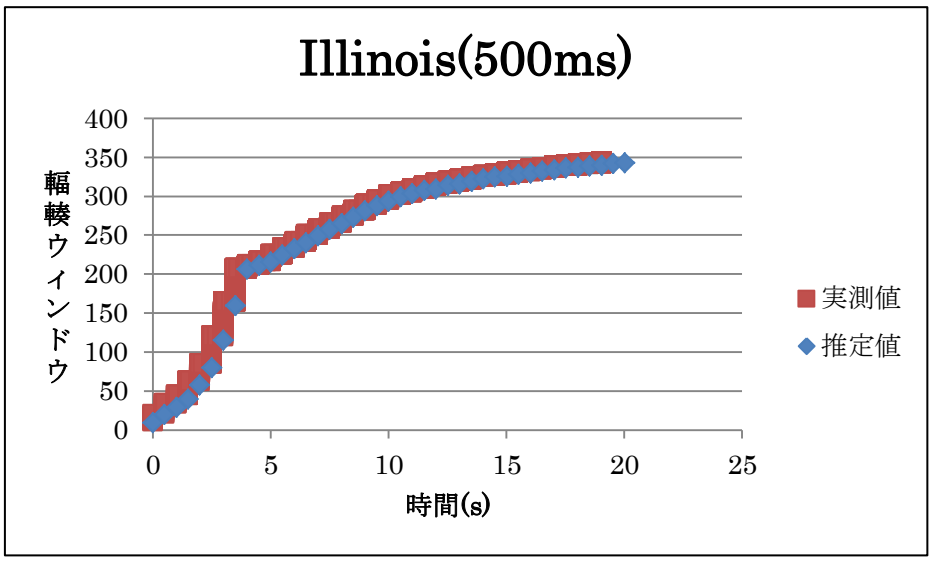

図 37. 500ms 遅延させたときの TCP Illinois の RTT 毎の輻輳ウィンドウの変化

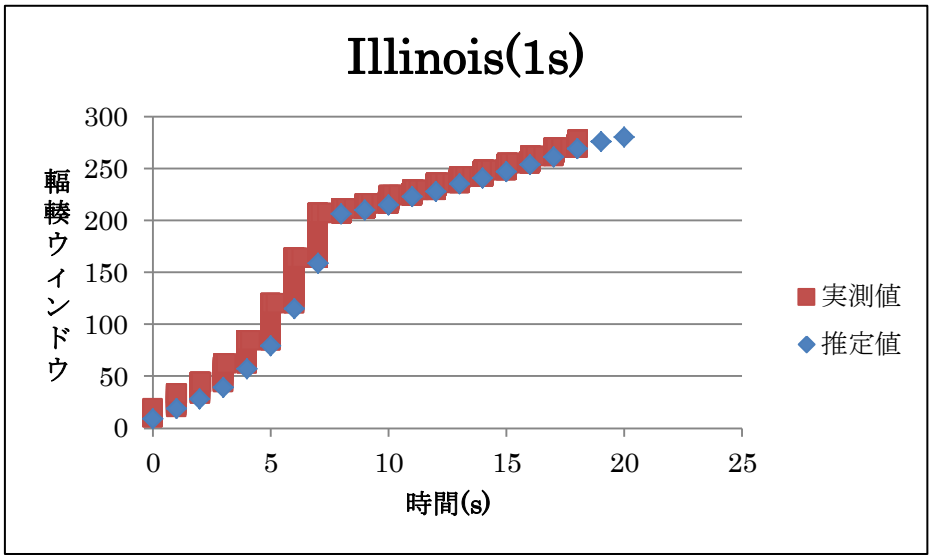

図 38.　1s 遅延させたときの TCP Illinois の RTT 毎の輻輳ウィンドウの変化

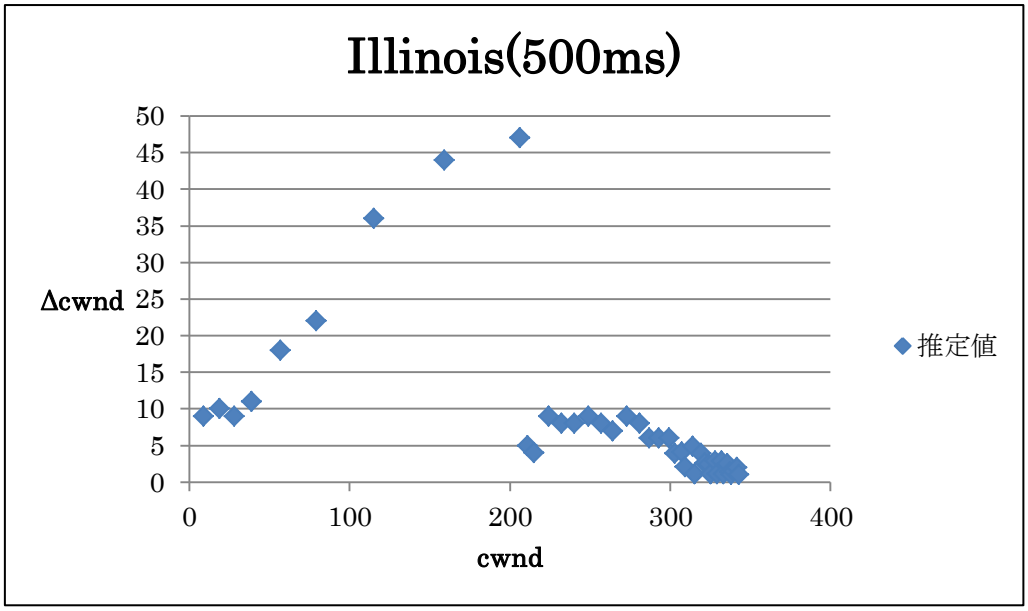

図 39.　500ms 遅延させたときの TCP Illinois の輻輳ウィンドウと変化量の推移

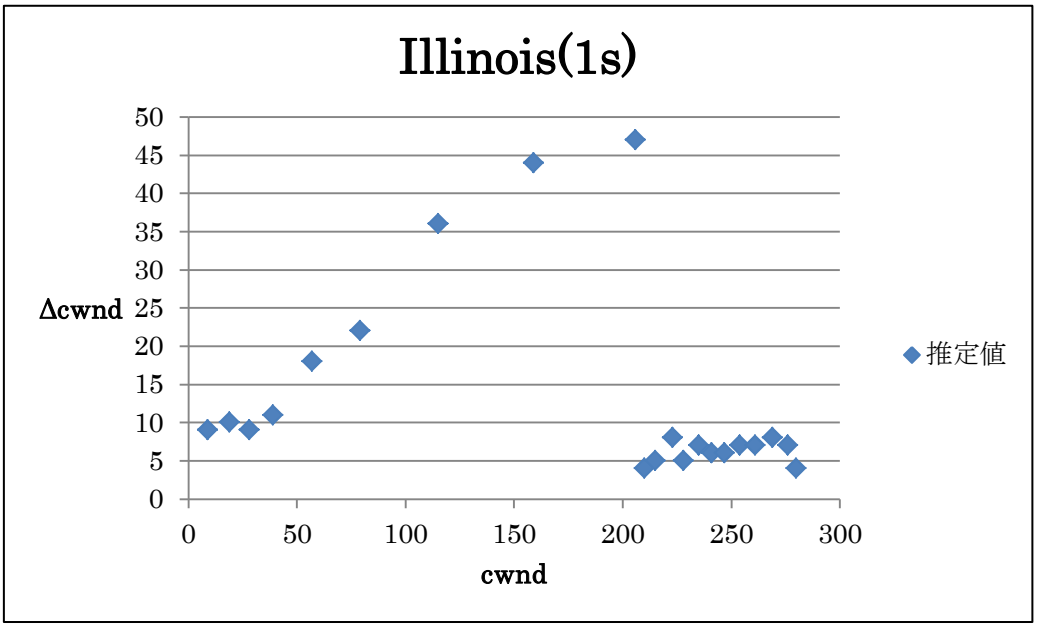

図 40．1s 遅延させたときの TCP Illinois の輻輳ウィンドウと変化量の推移

図 37，図 38 より，TCP Illinois の推定値は実測値に非常に近似しているため，推定値としての有用性は高いと考えられる．

また，図 39，図 40 の TCP Illinois のプロット推移は図 18 の散布予想図と比較すると，スロースタートフェーズにおいても輻輳回避フェーズにおいても非常に近似しているといえるため，有用性は高いと考えられる．

5.3　全体の評価

全体的に見ると，輻輳ウィンドウの推定値はどの TCP バージョンにおいても有効な値として使用できると考えられる．しかしながら，TCP バージョンの識別には，有用性は高いといえるものとそうでないものがある．Reno，Vegas は後者の結果になってしまった．Reno については，パケットロスが適切に起こらず輻輳ウィンドウの値がネットワークキャパシティの限界の数値を常駐していたからだと考えられる．Vegas についても同様で，パケットロスが起こらなかったため輻輳ウィンドウが下がらずに増加量

が 0 と 1 を行ったり来たりしていたのが原因だと考えられる．

また，TCP Reno，TCP Vegas，TCP Veno の輻輳ウィンドウの推定散布図が非常に似通ってしまっているため，現時点では識別が困難となってしまっている．

しかしながら，CUBIC と Illinois は輻輳ウィンドウの軌跡に特徴があり，推定値も散布予想図と近似した形をとっているため推定に有効な値として使用できると考えられる．

5.4　提案方式と評価に関する考察

本提案では輻輳ウィンドウの推定値を RTT 毎に分け，その時間内における最大値をとった．4.2 の手法における 2 の$cwnd_i$をそのまま全て抽出すると，図 41 のように輻輳ウィンドウが微小時間の間に大きく差が出てしまい，推定値としての精度は高くなくなってしまう．よって RTT 毎に最大値を取ることで推定値を線形状にし，実測値の形にできるだけ近いものとすることでこれを解決した．

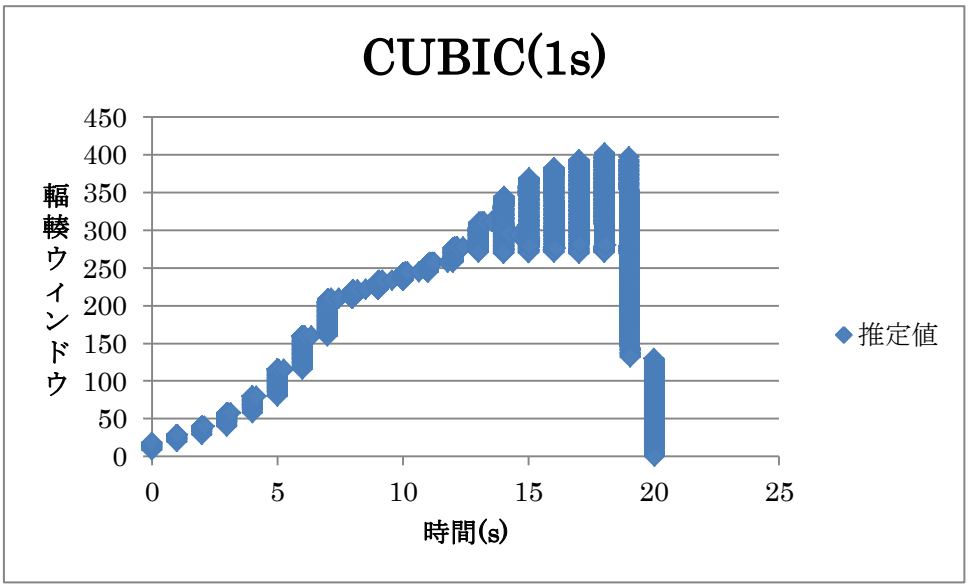

図 41．$cwnd_i$を全て抽出させたときの CUBIC TCP の輻輳ウィンドウの変化

第6章　　おわりに

6.1　まとめと今後の課題

本稿では，モニタリングから集めたアクセスログをもとに，輻輳ウィンドウと TCP バージョンの推定を行う手法を提案した．

そして RTT 単位で算出した輻輳ウィンドウの推定値は実測値と比べて非常に近似しているため，推定値として有用であることを示した．また，その推定値を利用した TCP バージョンの識別は，特徴的なものであれば推定値として使用できることを示した．

今後の課題として，今回使用した TCP バージョンは５種類と少なめだったため，TCP バージョンの種類を増やしていく必要がある．

また，今回はパケットロスが上手く起きなかったことが原因で識別が困難となってしまったものが存在するため，パケットロスを能動的に起こした実験を行い，推定が有用であると確定できるようなデータを収集する必要がある．

6.2　謝辞

本稿を作成するにあたって，多くの方々の指導と助言をいただきました．

特に主任指導教員である加藤聰彦教授には研究方針や進め方について人一倍多くの指導をいただきました．

また，策力木格助教授からも研究方針に関する助言をいただきました．

そしてエミュレーションの実験をするにあたって同研究室の大畑君にもお世話になりました．

改めましてここに深く感謝の意を示したいと思います．ありがとうございました．

参考文献

[1] W・リチャード・スティーヴンス, 詳解 TCP/IP Vol.1 プロトコル[新装版], 第 5 刷, 編 Pearson Education Japan, 2000.

[2] 加藤聰彦, コンピュータサイエンス教科書シリーズ 10 インターネット, 第 1 刷, コロナ社, 2012.

[3] Alexander Afanasyev, Neil Tilley, Peter Reiher, Leonard Kleinrock, "Host-to-Host Congestion Control for TCP," IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 12, NO. 3, THIRD QUARTER 2010.

[4] Injong Rhee, Lisong Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," In Proceedings of PFLDnet, 2005.

[5] Lawrence S. Brakmo, Larry L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet,” IEEE Journal on Selected Areas in Commun. 13(8), 1465-1480, 1995.

[6] Chengpeng Fu, Soung C. Liew, “Tcp Veno：TCP enhancement for transmission over wireless access networks,” IEEE Communications 21, 216-228, 2003.

[7] 徳田航一,長谷川剛,村田正幸, “HighSpeed TCP の性能評価とその性能改善方式の提案,” 信学技法,NS2002-302, March 2003.

[8] 武田稔,竹中豊文, "広帯域に対応した RTT ベース TCP 輻輳制御方式," 信学技法, NS2005-158, March 2006.

[9] Shao Liu, Tamer Bas¸ar and R. Srikant, “TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks”, First International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS), October, 2006.

[10] V. Paxson, “End-to-End Internet packet dynamics,” Proc. of ACM SIGCOMM ’97, 1999.

[11] S.Jaiswel, et. Al., “Inferring TCP Connection Characteristics Through Passive Measurements, “Proc. of INFOCOM 2004, March 2004.

[12] 大塩純平, 阿多信吾, 岡育生, “クラスター分析に基づく各種 TCP バージョンの識別手法,” 信学技報, NS2008-179, March 2009.

[13] 茂木重憲, 渡辺晶, “輻輳パラメータのリアルタイム推定,” 情報処理学会論文誌, Vol.52, No.3, pp.1308-1322, March 2011.

[14] P. Yang, et. Al., “TCP Congestion Avoidance Algorithm Identification,” Proc. of ICDCS’11, June 2011.

[15] “Wireshark · Go Deep.,” [オンライン], http://www.wireshark.org/.

[16] "tcp_testing | The Linux Foundation," [オンライン], http://www.linuxfoundation.org/collaborate/workgroups/networking/tcp_testing.